DESIGN AND DEVELOPMENT OF A TEMPORAL INFORMATION WAREHOUSE

VO THI NGOC CHAU

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTER ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2005

ISBN 974-15-1469-7

หัวข้อวิทยานิพนธ์	การออกแบบและพัฒนาระบบคลังข้อสนเทศเชิงเวลา
ผู้วิจัย	Vo Thi Ngoc Chau
รหัสนักศึกษา	46061024
ระดับการศึกษา	วิศวกรรมศาสตรมหาบัณฑิต
ภาควิชา	วิศวกรรมคอมพิวเตอร์
ปี	2005
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ. ดร. ศุภมิตร จิตตะยโศธร

บทคัดย่อ

ในช่วงหลายปีที่ผ่านมา ระบบสารสนเทศจำนวนมากได้ถูกพัฒนาขึ้นเพื่อรองรับงานระดับปฏิบัติการ การที่จะรวบรวมข้อมูลข้อสนเทศจากฐานข้อมูลของระบบงานระดับปฏิบัติการเหล่านี้เข้าด้วยกัน เพื่อนำเสนอให้แก่ระดับบริหารนั้นเป็นเรื่องจำเป็นและท้าทายอย่างยิ่ง ระบบคลังข้อสนเทศซึ่งเป็นระบบฐานข้อมูลเพื่อการบริหารและมีการเตรียมข้อมูลโดยการบูรณาการข้อมูลจากระบบระดับปฏิบัติการ จึงได้รับความนิยมอย่างกว้างขวาง ในปัจจุบันมีการนำปัจจัยเรื่องเวลามาพิจารณา โดยเฉพาะอย่างยิ่งในเรื่องการเปลี่ยนแปลงเพิ่มเติมตัวแอทริบิวท์ที่เป็นมิติซึ่งมีโอกาสเพิ่มขึ้นได้เมื่อเวลาผ่านไป อย่างไรก็ตามยังไม่มีการกำหนดหลักการออกแบบและพัฒนาระบบคลังข้อสนเทศเชิงเวลาซึ่งสามารถใช้โครงสร้างข้อมูลระดับแนวคิดร่วมกับฐานข้อมูลในระดับปฏิบัติการได้ วิทยานิพนธ์ฉบับนี้นำเสนอการออกแบบและพัฒนาระบบคลังข้อสนเทศเชิงเวลาโดยใช้แบบจำลองข้อมูลเชิงแนวคิดแบบไนแอม(ไออาร์เอ็ม) ซึ่งสามารถรับนิยามข้อมูลในรูปแบบของภาษากึ่งธรรมชาติ นอกจากนั้นยังได้นำเสนอ วิธีการเตรียมข้อมูลให้ครบถ้วนและวิธีการแปลงข้อมูลระหว่าง เวอร์ชั่นโดยใช้หลักการเชิงสถิติประกอบการทำคลัสเตอร์แบบค่าเฉลี่ยเค

Thesis Title	Design and Development of a Temporal Information Warehouse
Student	Miss. Vo Thi Ngoc Chau
Student ID.	46061024
Degree	Master of Engineering
Programme	Computer Engineering
Year	2005
Thesis Advisor	Assoc. Prof. Dr. Suphamit Chittayasothorn

ABSTRACT

A great number of operational systems have been built to support day-to-day operations for a long time. Integrating these operational systems is obviously necessary to achieve information of interest for managerial needs. Hence, data warehousing has been of growing interest so far for decision making supports. However, it is nowadays extended to temporal data warehousing with respect to time in order to deal with unavoidable problems existing in non-temporal data warehouses such as slowly changing dimensions, warehouse structural evolutions in their long serving duration of time. Unfortunately, there have been no standardized concepts on temporal data warehousing up to now. In addition, no full temporal data warehouse building cycle has been investigated using the cooperation of the conceptual design of a warehouse with the design of corresponding operational systems. Within such a context, our aim is designing a temporal information warehouse using the Nijssen's Information Analysis Methodology (NIAM) modeling, known as the Object Role Model (ORM) nowadays, as the data model both for user modeling of the warehouse and for the underlying implementation model of our system together with a deep-structured natural language interface. Further, we have developed two techniques based on the combination of the proportion notion and the K-means clustering technique for the enhancement of the temporal warehouse data availability at the extraction, transformation, and loading (ETL) process and at the transforming process among warehouse structure versions over time.

Acknowledgements

First of all, I would like to thank all people who have supported and helped me to obtain the scholarship from the AUN/SEED-net project so that I was offered a good opportunity to be here for my higher education and to approach interesting cultures, new knowledge, and good friends from other Southeast Asian countries.

Without a great deal of help and a great number of advices from my advisor, Assoc. Prof. Dr. Suphamit Chittayasothorn, this thesis wouldn't have been completed. I highly appreciate not only his time, but also his confidence he has put into me. He is very patient to listen to what I attempt to present during my research period. Furthermost, he is so kind that he always forgives me and gives me other good chances if I make mistakes or lose my ways. From my deep respects, I would like to express my thanks to him.

Also, all knowledge I achieved at this degree comes from many lecturers at Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand. They taught all of us, my friends and me, with their enthusiasms although some of courseworks I took during four semesters were not quite relevant to my research. At this moment, I might not take advantage of such courseworks as much as the lecturers expected. However, all courseworks helped me train my thinking and they will be necessary for my future work hopefully. For their willingness, I would like to thank all the lecturers.

And I would like to thank my former supervisor and other lecturers at Faculty of Information Technology, Ho Chi Minh City University of Technology, Vietnam, together with all of my friends including Vietnamese friends and non-Vietnamese friends, for their kindness and their encouragement they gave me directly as well as indirectly. They have stood by me with valuable advices and help.

To my Parents, my Sister and my Brothers! You are all in my heart.

Vo Thi Ngoc Chau

Contents

Contents (cont.)

Contents (cont.)

Contents (cont.)

List of Tables

List of Figures

List of Figures (cont.)

List of Figures (cont.)

List of Figures (cont.)

Chapter 1

Introduction

In recent years, data warehousing has come just in time as a popular choice of the integration of many various heterogeneous operational data sources for the knowledge discovery process in decision support systems. A data warehouse is formally defined as a collection of integrated, non-volatile, time-varying, and subject-oriented data from a number of autonomous data sources for decision making support [55].

However, apart from the omnipresence of the dimension time in data warehouse systems to track the evolution of fact data, a large number of proposed models have investigated time aspects in a data warehouse and extended a data warehouse to obtain a so-called temporal data warehouse. Temporal data warehousing has immediately been of growing interest for the solutions to dimension data evolutions known as slowly changing dimension data, on warehouse schema evolutions, and on dynamically changing sources in data warehousing [14]. Within a temporal data warehouse, the availability of data is significantly crucial for front-end tools such as online analytical processing tools, data mining tools, decision support systems, and so forth.

1.1 Literature Reviews on Temporal Data Warehousing

Many works on temporal data warehousing have been carried out so far.

Based on what were employed from the temporal database area, which levels, instance and schema, were taken into account, and the organization of warehouse structures in a warehouse system, researchers expressed their different views of time on a data warehouse. In general, these research investigations can be classified into several directions such as issues at the back-end side, in warehouses itself, and at the front-end side according to the data flow starting from operational sources, through warehouses, and ending at online analytical processing (OLAP) applications, information systems using data mining techniques, and decision support systems (DSSs) [55].

First of all, according to [2 and 10], time-stamping data should be considered at the loading task so that warehouse data will be temporal data with valid time or bi-temporal data with both transaction time and valid time.

[2] presented how to achieve time-stamped operational data for warehouse data with valid time as transaction time of data sources while [10] gave us a deeper study on all kinds of source temporal features. These authors organized their own data warehouse without any type of well-known warehouse schemas; that is, "dimension data" and "fact data" for aggregation operations are equally treated at this stage. Then, from the corporate data warehouse, defined as a bi-temporal database, data marts are built using some available warehouse schema to support front-end tools as usual. However, it is likely that they supposed other problems at the extraction, loading, transformation process such as noisy data, incomplete data, and so on were satisfied.

No attention on the loading process, several proposals [42, 15, 46, 3, 44, 19, 9, 31, 20, 21, 22, 32, 8, 37, 7, and 53] were supposed to work on post-loaded warehouse data. Warehouse data will be then reorganized according to their models for a so-called temporal data warehouse. Depending on which kinds of changes, dimension data or warehouse structures, time information is added at the instance or schema level or at both levels during the reorganization process.

Basic changes figured out in [30, 14, 31, 21, 22, and 32] comprise dimension data changes, schema changes, and operational constraint modifications. Up to which changes the proposed solutions focus on, the time-stamping is added at the instance level, or at the schema level, or at both levels. For the time-stamping at the instance level, their so-called temporal data warehouse supports state-oriented data and expresses the history of data sources responsible for warehouse data [42, 15, 46, 3, and 37]. For the time-stamping at the schema level, their temporal data warehouse becomes a multi-version data warehouse as introduced in [7, 53] where each version is stable at the instance level and at the schema level in an interval of time. An interval of time is a version life span. For the time-stamping at both instance and schema levels, a temporal data warehouse is able to cope with slowly changing dimensions as well as structural changes [19, 31, 20, 21, 22, 32, and 8].

Among these temporal data warehouse models [42, 15, 46, 3, 44, 19, 9, 31, 20, 21, 22, 32, 8, 37, 7, and 53], some of them just updated warehouse data to make them consistent with the current state of operational sources [30], some of them allowed a sequence of data versions to be kept consistently with the history of operational sources [42, 15, 46, 3, and 44], and the rest supported warehouse data of a structure version to be consistent in another structure after inter-version transformations [19, 31, 20, 21, 22, 32, 8, and 7]. In the proposals which took into account inter-versions transformations [19, 31, 20, 21, 22, 32, 8, and 7], linear functions in form of $f(x) = kx$ where x is a measure value of a source version mapped into a target one are provided. However the factor k called a weighting factor in [19, 20, 21, and 22] is determined by users or predetermined by their systems for uniformity transformations. For further information, [31 and 32] introduced confidence factors with these functions. These confidence factors are qualitative or quantitative. There are four values for qualitative factors, sd for source data, em for exact mapped data, am for approximated mapped data, and uk for unknown mapping. Even though [31 and 32] helped users define the factor k using a so-called global quality factor, users are again asked for their assistance. Because facts in a warehouse are huge and transformation functions are applied to each measure in every fact of a source version which is going to be transformed, user effort will increase in order to input lots of values for the factor k if they want more precise transformed data; otherwise, the transformation will be uniform. From their introduction to inter-version transformations [20, 21, and 22], uniformity is illustrated.

Further, there are some assumptions on these reviewed papers. Both warehouses and operational sources are formerly designed except for [55, 42, 15, 30, and 14]. Warehouse data are already loaded into the warehouse except for [55, 42, 15, 30, 14, 2, and 10].

For the implementation, [15, 2, and 10] built a temporal data warehouse using an object-oriented database management system (DBMS) while [42, 3, 7, and 53] using Oracle DBMS, [19, 20, 21, and 22] using Oracle DBMS for a temporal data warehouse and Hyperion Essbase for data marts, [31 and 32] using MSQL Server 2000 DBMS.

Summing up, tracking the evolutions not only of dimension data but also of warehouse structure is required to enhance the availability of warehouse data for

queries spanning over an interval of time in which the warehouse structure is unstable. From the review, the problem of keeping track of structural evolutions in a warehouse and the problem of inter-version transformations within an actual bi-temporal data warehouse are still promising. Besides, although how data come to a (non-temporal, temporal and bi-temporal) warehouse from operational sources also affects the availability of warehouse data, not much investigation is for this issue. The following tables are a summary on some references for a convenient comparison among them based on some points listed as follows:

- (a) Where time-stamping data is considered: before loaded into the warehouse (back-end), in the warehouse, during leaving the warehouse (front-end) if data flows are considered from operational data sources, through warehouses, to applications (OLAP, data mining, DSS, etc.)
- (b) Which kind of time information : valid time or transaction time or both
- (c) Which level is time-stamped : instance or schema or both
- (d) Warehouse model
- (e) Transformation
- (f) Note

Table 1.1 Summary on (a), (b) and (c)

Reference	Where (a)	Time Kind (b)	Level (c)
[42]	Warehouse at the logical design level	Transaction time for event-oriented data and valid time for state-oriented data, dimension data and fact data are equally treated	At the instance level for each row in every table
[15]	Warehouse at the conceptual design level	Valid time	At the instance level with three kinds of time-stamping, namely, set-of-classes, class, and attribute time-stamping

Table 1.1 (cont.)

Reference	Where (a)	Time Kind (b)	Level (c)
[46]	Warehouse	Transaction time	At the instance level for each row changed at the sources
[3]	Front-end	Valid time	At schema or instance or both levels
[44]	Warehouse	Transaction time and valid time	At the instance level with the time-stamping of the attribute level
[19]	Warehouse with respect to the front-end side	valid time	At both instance and schema levels
[9]	Warehouse	Valid time	At the dimension schema level for both dimension instance changes and dimension schema changes
[31]	Front-end	Valid time	At both instance and schema levels
[20], [21], [22]	Warehouse with respect to the front-end side	Valid time	At both instance and schema levels
[37]	Front-end	Valid time	At the instance level
[8]	Warehouse	Transaction time, valid time	At both instance and schema levels
[2]	Warehouse with respect to data sources	Transaction time and valid time	At the instance level for each attribute of a class within the object-oriented modeling of a warehouse
[7]	Warehouse with respect to data sources	Valid time	At the schema level

Table 1.1 (cont.)

Reference	Where (a)	Time Kind (b)	Level (c)
[10]	Warehouse with respect to data sources	Transaction time and valid time	At the instance level for each attribute of a class within the object-oriented modeling of a warehouse
[32]	Front-end	Valid time	At both instance and schema levels

Table 1. 2 Summary on (d), (e), and (f)

Ref.	Model (d)	Transformation (e)	Note (f)
[42]	A temporal star schema without the dimension time	No	Solve the problem of slowly changing dimensions. No consideration at the schema level; so, no transformation is mentioned
[15]	Temporal classes contain detailed data evolutions to represent the past states of entities while archive classes contain non-detailed data evolutions to aggregate temporal objects.	Not mentioned	Model a data warehouse using an object-oriented approach with temporal classes and archive classes over a sequence of discrete, linear points in time. A data warehouse is a temporal materialized view over a global data source obtained from many various heterogeneous sources.

Table 1.2 (cont.)

Ref.	Model (d)	Transformation (e)	Note (f)
[30]	From their point of view, warehouse schema evolutions adequately addressed stem from user requirements in OLAP applications, not from the schemas of operational data sources. An object-oriented framework is built for the management of warehouse schema evolutions at the conceptual level using their schema evolution algebra. Even though transformations at both schema and instance levels are proposed, the underlying warehouse structure is currently reflected, not along time.		
[46]	A warehouse is a set of materialized views, temporal views for time information specified by temporal relational algebra expressions and non-temporal view complements including source relations and information for the re-computation of all source relations. The history of a source is not completely stored by updating temporal views.	Although data source states are tracked, the warehouse structure evolution is in fact unsupported. Therefore, there is no transformation.	A temporal data warehouse is a warehouse supporting historic information for data source states.
[14]	The dynamicity of heterogeneous operational sources is their autonomy and their purpose independence of data warehouses that are built from these sources. These sources may change their data as well as their structures without regarding the data warehouses. Hence changes in operational sources need to be propagated to the data warehouses so that the definition and schema structure of these data warehouses might evolve and the content of the data warehouses also needs to be correspondingly adjusted. XML is made use of in wrappers affected by these changes (data, schema and constraint modification). In brief, the paper figured out why evolutions can exist in a warehouse and what the evolutions have a strong impact on. Using the state orientation, they considered temporal data for data warehouses to ensure that information discovered from the data warehouses is analytically processed in consistency. The attribute level time-stamping within a bi-temporal model is employed.		

Table 1.2 (cont.)

Ref.	Model (d)	Transformation (e)	Note (f)
[3]	A temporal multidimensional model contains dimensions whose elements are time-stamped at the schema or instance level (or both) to track the updates on dimensions. Except for the dimension Time, other dimensions become temporal dimensions associated with a level belonging to the dimension Time and all of dimension instances are time-stamped. And besides normal fact tables, the warehouse built using their proposed model contains temporal fact tables which are associated with measure data over a set of points in time.	No	A temporal multidimensional query language is invented for temporal OLAP queries to warehouses built using the temporal multidimensional model.
[32]	A temporal multidimensional model is developed by means of versions at both instance and schema levels to allow schema evolutions. A star schema as a logical model is extended with a multi-version fact table which is a fact table with integrated confidence factors, and a new dimension called temporal modes of presentation. Temporal modes of presentation are modes stemming from structure versions of a warehouse and the temporally consistent mode. Each structure version is valid in an interval and kept unchanged over its life span.	Mapping measure data among versions is carried out with linear functions with factors determined by a mapping relation table.	This paper is an extension to the previous one [31] with more explanation about notions that they mentioned in [31]. Specially, a global quality factor is calculated using their proposed formula, but with assistance from users.

Table 1.2 (cont.)

Ref.	Model (d)	Transformation (e)	Note (f)
[2]	A data warehouse is viewed as a bi-temporal database. Their model treats "dimension data" and "measure data" equally by means of Current/Historical tables as bi-temporal storage structures for a warehouse. They studied all kinds of data sources with respect to temporal support.	Inconsideration on any kinds of evolutions at both instance and schema levels	Once a data warehouse is considered as a bi-temporal database, transaction time is entirely maintained by the system; that is, transaction time in the data warehouse is always obtained by adding transaction time into data at the loading process while the obtaining of valid time for the warehouse depends on the temporal characteristics of the data sources from which data are loaded into the warehouse.
[7]	A data warehouse is a set of versions where each version is valid in a certain interval of time. The changes on both contents and structures in autonomous data sources might lead to warehouse schema changes. Therefore, these changes need to be correctly propagated to the data warehouse at both instance and schema levels.	Referred to some previous papers of theirs [19, 20, 21, 22]	Multi-version data warehouses are used for handling changes in the structure of their schemas as well as simulating alternative business scenarios. Even though many kinds of both data and schema evolutions are studied, only dimension data evolutions are presented. Moreover, no inter-version relationship in terms of data transformations is taken into consideration.

Table 1.2 (cont.)

Ref.	Model (d)	Transformation (e)	Note (f)
[9]	A data warehouse is regarded as a historical database since one of warehouse data characteristics is historic. Each dimension whose hierarchies are evolved is associated with a set of points in time to be written as an interval of time. At every point in this set, the dimension is non-temporal.	No	The effect of dimension changes on fact data is not considered. This leads to the fact that fact data appear to be associated with only the current consistent dimensions and non-evolved dimensions.
[10]	A data warehouse is viewed as a bi-temporal database. Their model treats "dimension data" and "measure data" equally by means of Current/Historical tables as bi-temporal storage structures for a warehouse. They studied all kinds of data sources with respect to temporal support.	Inconsideration on any kinds of evolutions at both instance and schema levels	This paper seems to be an extension to the previous one [2] with explanation about all kinds of data sources with respect to time aspects.
[19]	A temporal data warehouse with structure data including dimension data and hierarchical structures which are time-stamped, structural mapping data including transformation functions, and fact data. Not only the history of source data, but also the history of warehouse data is taken into account.	Uniformity transformations or transformation functions with user-defined weighting factors to split each fact data into some sub-fact data	Even though, time information is added at the schema level, this paper considered the problem of slowly changing dimensions. Anyway, it is extensible to the problem of warehouse schema evolutions.

Table 1.2 (cont.)

Ref.	Model (d)	Transformation (e)	Note (f)
[31]	A temporal multidimensional model is developed by means of versions at both instance and schema levels to allow schema evolutions. A star schema as a logical model is extended with a multi-version fact table which is a fact table with integrated confidence factors, and a new dimension called temporal modes of presentation. Temporal modes of presentation are modes stemming from structure versions of a warehouse and the temporally consistent mode. Each structure version is valid in an interval of time and kept unchanged over its life span.	Mapping functions with confidence factors are used for mapping measure data among versions. These confidence factors are quantitative or qualitative.	Mapping functions and qualitative confidence factors need to be studied more for mapping measure data among schema structure versions. There is no explanation about the temporally consistent mode. From my understanding, the temporally consistent mode is a mode in which dimension data and fact data are consistent in the interval of time for this mode after mappings from actual modes that contain the data. This mode is specified by users in an interval of time.
[44]	A bi-temporal model is employed with the sequence of changes to the organizational history of sources at the point in loading time.	No	Based on the state orientation, they considered temporal data for warehouses to ensure that information discovered from the warehouses is analytically processed in consistency with source data changes.

Table 1.2 (cont.)

Ref.	Model (d)	Transformation (e)	Note (f)
[37]	Temporal data warehouses reflect the evolving history of an enterprise; particularly the history of dimension data changes and dimension hierarchies. An OLAP server using this model of a temporal data warehouse, called a TOLAP, works on the warehouse with queries specified within the interval of valid time for each valid time dimension by users.	No	To keep track of dimension data evolutions, each dimension is time-validated and called a valid time dimension. Further, each edge in hierarchies in a valid time dimension is associated with an interval of time.
[20], [21], [22]	A data warehouse allows all modifications on data as well on structures on a linear time axis in terms of structure versions. A structure version is a view on a temporal data warehouse valid for a given time period. Within a structure version, the structure of dimension data is stable at both schema and instance levels. Further the COMET meta-model is used to deal with modifications on warehouse structures.	Transformation functions with weighting factors stated by the administrator for converting data among structure versions on the axis time	They developed a temporal data warehouse from a data warehouse with the time-stamping of both data and structures. Transformation functions are provided to allow OLAP applications to consistently analyze data from many various structure versions within a structure version over time.

Table 1.2 (cont.)

Ref.	Model (d)	Transformation (e)	Note (f)
[8]	A bi-temporal data warehouse model is built with a bi-temporal extension of the meta-model that manages the time-stamping of all structural and data items in a bi-temporal warehouse. Valid time represents changes of fact data and structural data of dimensions, transaction time reconstructs the data that led to a particular decision using the knowledge stored in the warehouse at a given point in time.	Referred to some of their previous papers [19, 20, 21, 22]	It is useful to explain the role of transaction time in a warehouse and show how to apply the transaction time concept in a warehouse. It is quite theoretical to propose such a bi-temporal data warehouse.

Furthermore, an extension to the SQL language was proposed in [53] to make queries in multi-version data warehouses. A data warehouse with many versions is created to handle changes in external data sources from which data are loaded into the data warehouse. A data warehouse administrator creates a version of a data warehouse explicitly and this version represents the structure and content of external data sources within a given period of time. From their point of view on their approach to multi-version data warehouses, there is no need of excessive transformations of data in contrast to some previous temporal approaches [19, 20, 21, and 22].

This paper [53] seems to be an extension to the previous one [20] with querying multi-version data using their proposed SQL-based language. They enhanced queries to be executed on a current warehouse version, on a set of real warehouse versions, and on a set of alternative warehouse versions which are used for what-if analysis. For the first case, there is no extension to the SQL query language. However, a trade-off of common data shared by warehouse versions is the collection of all data which logically belong to the current one. For the second case, valid time for each version is used for selecting versions in which users are interested. They decompose the query into the set

of partial queries, each of which is classified for an appropriate version based on the version valid time. For the last one, because all intervals of valid time of alternative versions are limited to the utmost parent real version and linearly unordered, they require users to explicitly specify a set of alternative versions of interest. Up to requirements from users, partial results need to be merged into one result set by users' specifying in their queries using transformation methods. Unfortunately, there is no explanation about mentioned transformation methods. Case study is demonstrated with only changes of dimension instance structures and the addition of a level into a dimension at the upper position which doesn't require any inter-version transformations. This case normally causes no changes in facts which are associated with the affected dimension.

1.2 Problem Statements

Starting from all literature reviews related to temporal data warehouses, interesting problems are identified for the design and development of a temporal information warehouse both from potential problems in data warehousing such as how to integrate operational data from various sources into a common model, how to handle source changes and make these changes relevant to a warehouse, how to manage warehouse schema evolutions [27], how to handle time in data warehouses [35], etc. and from partially solved problems under some assumptions in temporal data warehousing. Each problem will be carefully examined again with as many existing proposals as able to be reached. Hence, each of our proposed solutions will be able to be well evaluated in comparison with respective solutions from those proposals.

The first is the problem of warehouse schema evolutions together with inter-version transformations. The problem is not quite new, but it is significant to improve transformed measure data among warehouse structure versions so that users are able to analyze their data over time if they are allowed to see how their organizational structure changes in a few periods of time. Managing schema evolutions is of course taken into consideration.

The second is the problem of missing data at the cleaning phase in the warehouse building cycle. Only [2, and 10] paid attention to how data are moved from operational

data sources into a warehouse; and the others [42, 15, 46, 3, 44, 19, 9, 31, 20, 21, 22, 32, 37, and 7] supposed data are already available in the warehouse which is being considered. For their work, the temporal issue is mainly focused with the assumption that all other subtasks at the extraction, transformation, loading (ETL) process are well satisfied. In other words, there has been little effort on the ETL process in a temporal data warehouse system so far. Unfortunately, the warehouse data quality is obviously affected by the subtasks at the ETL process. One of these subtasks is cleaning data before operational data are put into a warehouse. Hence, the handling of missing data is an interesting problem to enhance the data availability in a warehouse system.

The third is the design of a warehouse in cooperation with the design of operational sources at the conceptual level. This problem is derived from the assumption that the warehouse structure is formerly designed [46, 3, 44, 19, 9, 31, 20, 21, 22, 32, 37, and 7].

1.3 Existing and Proposed Solutions

This section will briefly present existing solutions and our proposed solutions for each of three main problems addressed above so that our contributions will be convincingly stated with achieved results in the following chapters.

1.3.1 Inter-Version Transformations for Warehouse Schema Evolutions

Basically, measure changes in fact tables are recorded in data warehouses and sophisticated tools are capable of analyzing data along time and other dimensions. However, current data warehouses are not well prepared for data changes in dimension tables as well as changes in a data warehouse structure in spite of their requirement for serving as a long term memory [20]. In order to avoid erroneous query results from existing data warehouses as much as possible, there are many methodologies proposed for developing data warehouses into temporal data warehouses by means of temporal database [35]. Extending temporal database concepts for data warehousing, lots of solutions have been investigated for providing interoperability and data access across evolving data warehouses, now known as temporal data warehouses. [32], [22], [20], and [42] are selected for the comparison of our solution to inter-version transformations with theirs in temporal data warehouses.

From the idea in temporal databases, most of the things that we observe change in time and the database system should capture the changes. This resembles one of the main characteristics of data warehouses – the data stored in a data warehouse is historical and time-dependent [42]. From this point of view, [42] suggested a so-called temporal star schema that differs from a traditional star schema in its treatment of time. The dimension Time is removed from the temporal star schema and instead each row in every table of the schema is timestamped. Therefore, it treats the fact table and the dimension tables equally with respect to time. Although the problems of data changes in dimension tables are disappeared using temporal star schemata, their temporal warehouse is not built to handle changes in the warehouse structure, and there is thus no data comparison and data conversion over time.

Another approach to a temporal data warehouse is that [22] extended a regular data warehouse with three aspects including temporal extension, structure versions, and transformation functions. By temporal extension, dimension data have to be time stamped in order to represent their valid time. The valid time represents the time when a "fact is true in the modeled reality" [12]. Next, structure versions are used to track warehouse structure evolving over time as well as for coping with modifications of dimension data. And, transformation functions are provided to transform data from one structure into another structure. The combination of structure versions and transformation functions enables us to handle data changes in dimension tables and changes in warehouse structures, as well [22]. To support transformation among warehouse structure versions, weighting factors are introduced as a weight of a measure considered in transforming. For each transformation function, a weighting factor is stated by the administrator or calculated equally as all the weighting factors of transformation functions from quarterly data to monthly data are 1/3 [20].

Also using concepts of structure versions and mapping functions, [32] showed a temporal data warehouse containing the temporal multidimensional schema, and metadata related to it, including mapping relationships in a temporal multidimensional model. Instead of weighting factors in [20], their mapping uses so-called confidence factors which describe the reliability of data and distinguish source from mapped data with sd that stands for source data or temporally consistent data, em for exact mapped

data, am for approximated mapped data, and uk for unknown mapping relationships. These factors are given by users. It means that the conversion between two versions is not automatically carried out. By getting factors such as weighting factors and confidence factors as input, [32] and [20] can not re-compute values from one structure version into another after adding/removing a new dimension or dimension level automatically because the number of factors increases in direct ratio to the number of transformation functions or mapping relationships. The bigger temporal data warehouse, the more factors we need for transforming or mapping. Unfortunately, these factors play a very important role in transforming to make mapped data of a structure version become more realistic data in the different warehouse structure version which the data were not loaded into and never belonged to before mapped.

Instead of weighting factors and confidence factors, probabilities are used in our proposed technique for the transforming of measure data from a version to another one. The technique will be automatically performed on source measure data and their associated probabilities that are calculated by using one of two provided choices. The first choice of the proportion calculation uses an individual period data sample. The second one uses a multiple period data sample. With a multiple period data sample, a single period average data sample is derived by employing the K-means clustering technique. Probabilities are computed using the final data sample. This technique combines the proportion notion and the K-means clustering technique to obtain as appropriate probabilities as possible for more accurate transformed measure data. Finally, our proposed technique reached the goal.

1.3.2 Missing Data

Handling missing data is a popular problem in the statistical data preprocessing area, the data mining area, and the data warehouse area. At present, as clearly reviewed in [4], a large number of methods for solving this problem such as ignoring missing data, clustering with incomplete data sets, filling in missing data using K-Nearest Neighbour [16 and 17], regression trees, and so forth have been proposed. Also, nine approaches to missing attribute values briefly presented and compared in [26] were well applied in the data mining area.

However, how to treat this problem has not been standardized in the data warehouse area although it is well-known that the cleansing phase is one of major tasks in the warehouse building cycle. Moreover, several available tools for the extraction, transformation, loading (ETL) process in data warehouse environments such as ARKTOS [41], AJAX, and some components integrated into some database management systems (DBMSs) such as Oracle Data Warehouse Builder and Microsoft SQL Server 2000 Data Transformation Services (DTSs) do not support users to cope with this problem. Instead, they replace the missing cells with some pre-defined values or some special symbols [5].

Realizing the shortage of consideration on the solution to this subtask at the cleansing phase in data warehouses, [28] and [39] paid attention to cleaning data from many heterogeneous sources before data are loaded into the warehouse. The linear regression in [39] and the use of average values in [28] were investigated for the filling of incomplete warehouse data. We found out that these two techniques faced difficulties with the assumption on data distributions. [39] required data to be able to be rearranged along the axis Time. This need leads to ignoring other dimensions associated with the fact whose measure data are being processed. As for [28], the uniform distribution is implicitly supposed to join their filling process.

In such a context, one of our aims at the ETL process is to improve estimated measure data for missing-value cells in fact tables of a warehouse so that high-level managers will get complete data for their reports and make better decisions and predictions. To do that, we employ the K-means clustering technique with the combination of probabilistics and study the Euclidean distance for the evaluation on our technique.

1.3.3 The Design of a Warehouse

The aim of designing a warehouse is how to well organize warehouse data and make them available for front-end tools such as online analytical processing (OLAP) tools, data mining tools, reporting facilities, and so on. What should be represented in a multidimensional scheme has not been standardized up to now although many works on the physical/logical/conceptual design of a warehouse have been proposed so far [1].

Due to the popularity, a (multi) star schema is widely utilized as the structure of detailed-level data in a relational warehouse at the logical design level. The achievement of a (multi) star schema stems from operational schemas and managerial needs. These needs are normally captured for online analytical processing (OLAP) applications. Therefore, several well-known data models such as the Entity Relationship model (ERM) and the Unified Modeling Language (UML) have been employed for the conceptual design of a warehouse.

One of the first research works on a graphical conceptual model built using the ERM, [33] proposed a Dimensional Fact model semi-automatically derived from the existing ERMs of data sources. The Dimensional Fact model is a collection of tree-structured fact schemes considered as trees rooted by facts. With basic elements like facts, dimensions, attributes, and hierarchies, fact schemes present the additivity of fact attributes along dimensions, and the optionality of dimension attributes.

Similar to [33], [11] invented a so-called Multidimensional Entity Relationship (ME/R) model that is a multidimensional extension to ERM for conceptual modeling of OLAP applications in terms of facts, dimensions, and dimension levels. The ME/R is defined in graphical notations with a specialized entity set for dimension levels and specialized relationship sets containing a special n-ary relationship set for the 'fact' relationship set and a special binary relationship set for the 'rolls-up to' relationship set. These sets are used to capture the static structure of a multidimensional application domain. Besides, the mapping of the ME/R model to logical multidimensional data models is required.

Another proposal on a starER model also obtained from the combination of the star structure and the ERM together with special types of relationships supporting hierarchies is given in [36]. The new model is composed of four modeling elements including fact sets which show actual facts of the real world application environment, entity sets which present autonomous objects of the environment, relationship sets which describe associations or links among entity sets or among entity sets and fact sets, and attributes which show characteristics or properties of entity sets, relationship sets, and fact sets.

Further, an event-entity-relationship (EVER) model in [29] based on an event concept combined with an ERM is proposed for multidimensional modeling where

events are used to present measurement facts and entities are used for dimension categorizations. Instead of a graphical notation, a (multi) star schema is determined by so-called star path expressions in an entity-based model language. Moreover, [29] determined the temporal semantics of EVER as well.

Besides the use of the ERM, lots of efforts have been made for the multidimensional modeling within object orientation based on the UML [48, 51, 49, 50, 25, and 24]. The UML is extended with a set of stereotypes to represent main structural and dynamic multidimensional properties such as relationships between facts and dimensions at the conceptual level. A warehouse schema with facts and dimensions at the logical level is formed by classes Fact with fact attributes FactAttribute and classes Dimension with dimension attributes DimensionAttribute. A fact class is specified as a composite class in shared aggregation relationships of n dimension classes [49]. Furthermore, the Object Constraint Language (OCL) is proposed to specify the constraints attached to the defined stereotypes to avoid using these stereotypes arbitrarily. Aso, they well explained why the object-oriented approach was suitable for the modelling of warehouses [48] as well as what benefits were obtained from the use of the UML in the whole of warehouse design [51, 50, 25, and 24]. Even though their achievements have been valuable so far, users need to be trained with their approach so that they may get familiar with new graphical notations and make use of this model for building their warehouse. Further, they also take advantage of the UML to model the extraction, transformation, and loading (ETL) process, but how legacy sources organized operational data is likely to be ignored.

In brief, the approaches above are initiative to the design of a warehouse using new graphical notations. They derived a conceptual warehouse schema from an operational conceptual schema [33, 43], or from user requirements in OLAP applications [48, 51, 49, 50, 11, 25, 24, and 36]. This conceptual schema is transformed into a warehouse schema automatically [29, 49, 11] or semi automatically [33, 48]. Hence, it acts as a mediator between an operational conceptual schema and a respective warehouse schema.

Apart from the warehouse design based on the ERM and the UML, data warehousing from an ORM perspective was introduced in [52]. However, the potential

ability of the ORM to the construction of a warehouse system was manually exploited with an operational relational schema as an intermediate schema before an expected warehouse schema is reached.

Since it makes good sense to take into account not only the business needs in OLAP applications but also the modeling of operational sources from which data are put into a warehouse, this section focuses on a novel approach to designing a multi-star/star warehouse schema automatically and directly from an operational NIAM conceptual schema. This approach will have a good effect on gathering metadata of operational sources for establishing mappings between legacy sources and data warehouses at the ETL process. Furthermore, the NIAM modeling will be a choice of both the user modeling of warehouses and the underlying modeling of our system as earlier stated. It is due to the fact that the NIAM modeling is based on a fact-oriented approach and tables in the fifth normal form (5NF) can be obtained from a NIAM conceptual schema by using the Relational Mapping Procedure in [52] (it is the Optimal Normal Form (ONF) algorithm in [18]), and a large number of operational databases have been well designed in the NIAM modeling up to now. Above all, instead of new invented graphical notations [33, 48, 51, 49, 50, 11, and 24] and entity-based model languages [29], a deep-structure natural language interface is designed in our system so that users can conceptualize their requirements as well as represent Universe of Discourse (UoD) easily and expressively.

In summary, this research is going to focus on the achievement of a temporal information warehouse with schema versioning designed in the NIAM/ORM modeling through a simple deep-structured natural language interface. The design of a temporal information warehouse will be discussed within a full temporal data warehouse building cycle in cooperation with the design of operational databases. The obtained temporal warehouse data will be significantly available with no care about warehouse schema evolutions as well as about missing data from operational sources.

1.4 Assumptions

Due to the use of the proportion notion, the two proposed computational techniques ask users for sample measure data user-dependently although the degree of

dependence on user inputs is reduced using the K-means clustering technique. This interaction requires users to make sure that what they provide our system is from their best efforts.

Besides, only the cleaning phase is a key subtask in the extraction, transformation, and loading (ETL) process to be tackled. Therefore, it is assumed that all other subtasks in the ETL process are thoroughly satisfied.

Moreover, all dimension data will be stable in their own warehouse version owing to our main consideration on schema evolutions.

1.5 The Organization of this Thesis

This thesis is organized as follows.

Theoretical backgrounds on information warehousing, definitions in the temporal database field, the proportion notion, and the K-means clustering technique will be given in the chapter 2. The purpose of the chapter 2 is to provide several basic concepts for easily approaching our research work.

The chapter 3 will explain our proposed temporal information warehouse architecture, called the TIWA1 architecture, in details. This architecture is developed from a typical three-tier architecture of a regular data warehouse using the NIAM/ORM modeling. We will take advantage of such an architecture to design a temporal information warehouse through a deep-structured natural language interface.

We will pay attention to the ETL process in the chapter 4. This chapter will show how the back end side of a warehouse based on the NIAM/ORM modeling is considered. As far as we know, there are many sub-tasks in the ETL process. One of them is cleansing operational data from many legacy sources. This sub-task aims at removing noisy, inconsistent data as much as possible, at handling missing data, etc. so that data will be ready to be put into a warehouse. Our choice of the back-end sub-tasks is the treatment of missing data because few works have been proposed for filling in missing data.

After consideration on the ETL process, we go further to the design of a warehouse itself. The chapter 5 will introduce our approach to designing a warehouse with a

NIAM/ORM context free grammar. How to make use of the NIAM/ORM modeling of operational sources in a warehouse building cycle is also explained in this chapter.

Once operational data are well organized in the structure of a warehouse that allows schema versioning, needs on querying warehouse data over a long time are discussed in the chapter 6. Representing warehouse schema evolutions to adapt a current warehouse structure to the latest organizational conditions is carried out at the conceptual level with a deep-structured natural language interface so that we can put the same methodology of the design of a warehouse in the previous chapter to good use. The proposed technique for data transformations among warehouse versions is minutely presented in comparison with other classical techniques. The inter-version transformation will make all warehouse data consistent and available in one user-specified structure version in order that some existing front-end tool is able to discover as much information of interest as possible for reporting, decisions and predictions.

Next, the chapter 7 will present the implementation of our system to develop a temporal information warehouse with all functions on which this research work focuses. The implementation is carried out using Visual Basic .Net and Microsoft SQL Server 2000.

Finally, conclusions and future researches are pointed out in the chapter 8.

Besides these chapters, a few appendices are added for the user manual through user interfaces, for the explanation of test procedures to justify our solutions in temporal data warehousing, and for case studies.

Chapter 2

Theoretical Backgrounds

Due to the fact that temporal data warehousing brings two fields, namely data warehousing and temporal databases together, knowledge on both fields needs to be studied.

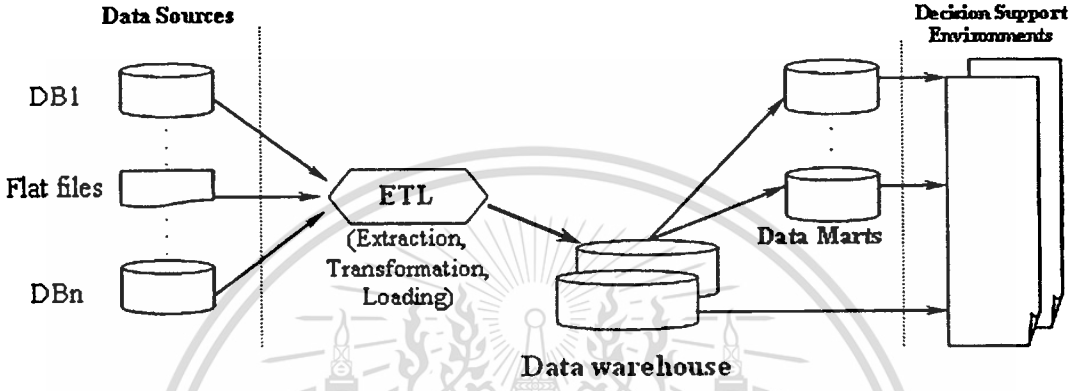


Figure 2.1 A Typical 3-tier Data Warehouse Architecture

The term "information warehouse" was firstly introduced in the IBM systems journal for reporting and data analysis from operational databases [6]. A few years later, W.H.Inmon, known as Father of Data Warehouse, formally coined "data warehouse" in [55]. By W. H. Inmon, a data warehouse is defined as a collection of integrated, non-volatile, time-varying and subject-oriented data from many heterogeneous legacy sources for managerial needs. These characteristics of warehouse data make a data warehouse be a special kind of databases. Firstly, the "integrated" feature of warehouse data requires all data inconsistencies from the heterogeneity of many various sources to be removed before the loading process. Secondly, once loaded into a warehouse, warehouse data are read-only. In other words, no updates of data are allowed in the warehouse environment. This mechanism leads to the "non-volatile" property of warehouse data. Thirdly, warehouses normally tend to have a serving time horizon longer than operational databases do. Moreover, warehouse data evolve over time to become historical. Finally, warehouse data are subject-oriented because data warehouses are developed with the aim at several major subject areas of the organization defined in the data model. A typical three tier architecture of a warehouse

in Figure 2.1 presents the data flow through a warehouse, from operational sources to front-end tools in decision support environments [45].

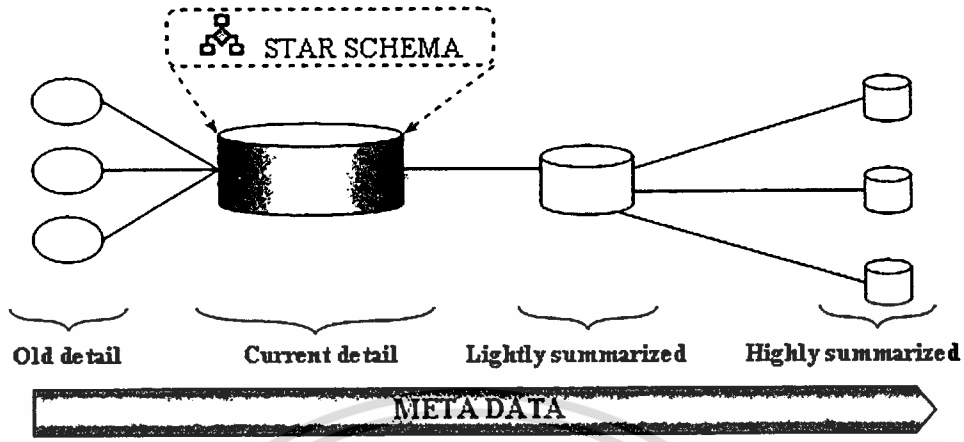


Figure 2.2 A Star Schema for the Warehouse Data Structure at the Current Detail Level

Especially, from 1994 up to 1995, the term "star join" was of interest in the database area. [47] presented the join among a large table and small tables for decision support databases. This kind of "join" was defined as cross products, also known as a star join. After that, [40] is regarded as the original proposal on "star join" with a formal definition and clearer details in which a very large table is called a detail table, and other smaller tables are called descriptive tables. At the same time, Ralph Kimball, known as Doctor of Data Warehouse, introduced a so-called star join schema at his site http://www.dbmsmag.com/9509d05.html. Also, the white paper "Star Schemas and STARjoin technology" from Red Brick Systems was published at their site at that moment. Day by day, a star schema is widely used for the warehouse data structure at the current detailed level in a data warehouse [55] as shown in Figure 2.2.

Technically speaking, a star schema is formed by a fact structure with one numeric attribute known as a measure, and several dimensions surrounding the fact structure. Most of the attributes of these dimensions are textual for the descriptive purposes. The key of a fact structure is composed of many attributes each of which refers to the key of a dimension as a foreign key. In addition, each dimension is commonly organized in terms of hierarchies which are relationships among dimension levels in the dimension. As an extension, a multi-star schema is made up by many fact structures and many dimensions related to each fact structure. These dimensions may be shared by several fact structures in a warehouse. To design a warehouse schema is determine which fact

structures are, which dimensions are associated with each fact structure, and which all hierarchies are in each dimension.

An illustration in Figure 2.3 is used as follows to show a simple star schema including a fact structure SALE and three dimensions CUSTOMER, TIME, and PRODUCT. The measure Sale in the fact structure SALE is the total number of products that a customer bought at a point in time. Each dimension has their own hierarchies described by arrows. A dimension level in a dimension hierarchy is known as a lower level if it is located at the root of an arrow, and vice versa, known as a higher level if it is located at the other end of an arrow. For example, the sequence TKey → Month → Quarter → Year is a hierarchy in the dimension TIME where TKey is the lowest level, and Year is the highest.

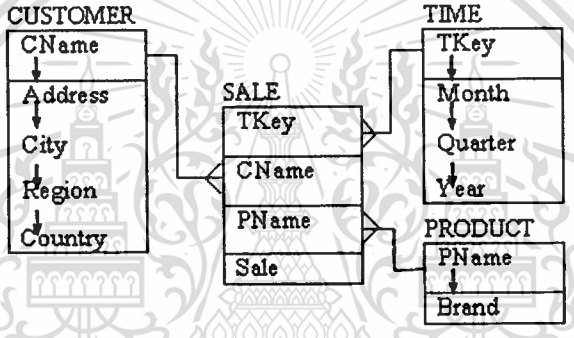


Figure 2.3 A Simple Star Schema

As for the contribution of the temporal database field in temporal data warehousing, the temporal nature of data was investigated in terms of "valid time" and "transaction time" where "the valid time of a fact is the time when the fact is true in the modeled reality and valid times are usually supplied by the user" and "the transaction time of a database fact is the time when the fact is current in the database and may be retrieved". Up to temporal aspects in a database, a temporal database is a database that supports valid time or transaction time, not counting user-defined time. Two other concepts important to be mastered for the development of temporal data warehouses are "schema evolution" and "schema versioning". The first is that "a database system supports schema evolution if it permits modification of the database schema without the loss of extant data". The latter is that "a database system accommodates schema versioning if it allows the querying of all data, both retrospectively and prospectively,

through user definable version interfaces". For more information on temporal database concepts, [12] may be referred.

Apart from data warehouses and temporal databases, mathematics plays an important role in this research. So, probabilistics and statistics were studied because our two proposed techniques for the treatment of missing data and for the inter-version transformation use the proportion notion. Proportion is simply defined as the ratio of the number of favorable cases to the total number of all cases under the condition that all the cases (chances) are equally probable.

In addition to probabilistics for techniques, we make the most of one of the most popular clustering techniques, the K-means clustering technique, in the well known field – data mining. Based on the capability of maximizing the intra-cluster similarity and minimizing the inter-cluster similarity of the K-means clustering technique [23 and 34], the calculation of probabilities was strongly supported so that estimated results from these two techniques were found to be more precise.

Besides, in order to design a warehouse in general, the Nijssen's Information Analysis Methodology (NIAM) modeling [18], known as the Object Role Modeling (ORM) [52] nowadays, was applied. The NIAM modeling is very popular all over the world because of the fact-oriented approach of the modeling and the ability of generating tables in the fifth normal form from a NIAM conceptual schema. Hence, it is very effective to design operational databases using the NIAM modeling. Also, it is very convenient to develop a deep-structured natural language interface for the modeling of a warehouse and it is very expressive for users to conceptualize their requirements with such an interface.

In brief, this research work on temporal data warehousing demands the study on data warehousing, on temporal databases, on the proportion theory and statistics, on clustering techniques, and on the conceptual modeling.

Chapter 3

The Proposed Temporal Information Warehouse Architecture

3.1 Concepts in our Temporal Information Warehouses

Before going into details, we regard a temporal information warehouse in terms of versions as [32, and 7] under the current circumstance that there have been no standardized concepts of a temporal data warehouse up to now whereas a lot of models have been invented for a temporal data warehouse. A temporal information warehouse is formally defined as an information warehouse with warehouse schema versioning where schema versioning allows the querying of all data, both retrospectively and prospectively [12]. In other words, a temporal information warehouse is a set of warehouse structure versions each of which is valid in a specific time interval where a time interval is the time between two instants and an instant is a time point on an underlying time axis [12]. All data and structural elements of a version were kept unchanged at both the instance level and the schema level. Each of structural changes with structural operations like adding/removing a dimension, adding/removing a dimension level leads to a new warehouse structure version. Multistar/star schemas are used for the warehouse data structure versions at the current detailed level [56] where a star schema is made up by a fact structure at the center and many dimensions surrounding the fact structure via the referential relationships, and a multistar schema is an extension to a star schema with many fact structures and many dimensions that are allowed to be shared among these fact structures.

Based on such a definition of a temporal information warehouse, the proposed temporal information warehouse architecture will be the style in which a temporal information warehouse is going to be constructed in this research work.

3.2 The Temporal Information Warehouse Architecture (TIWA1)

Developed from a typical data warehouse architecture in Figure 2.1, our proposed temporal information warehouse architecture, called the TIWA1 architecture, includes

the back-end side, the warehouse itself, and the front-end side to represent the flow of data from legacy sources to end-users in decision support environments. Following the flow of data allows us to design a temporal information warehouse within a full warehouse building cycle.

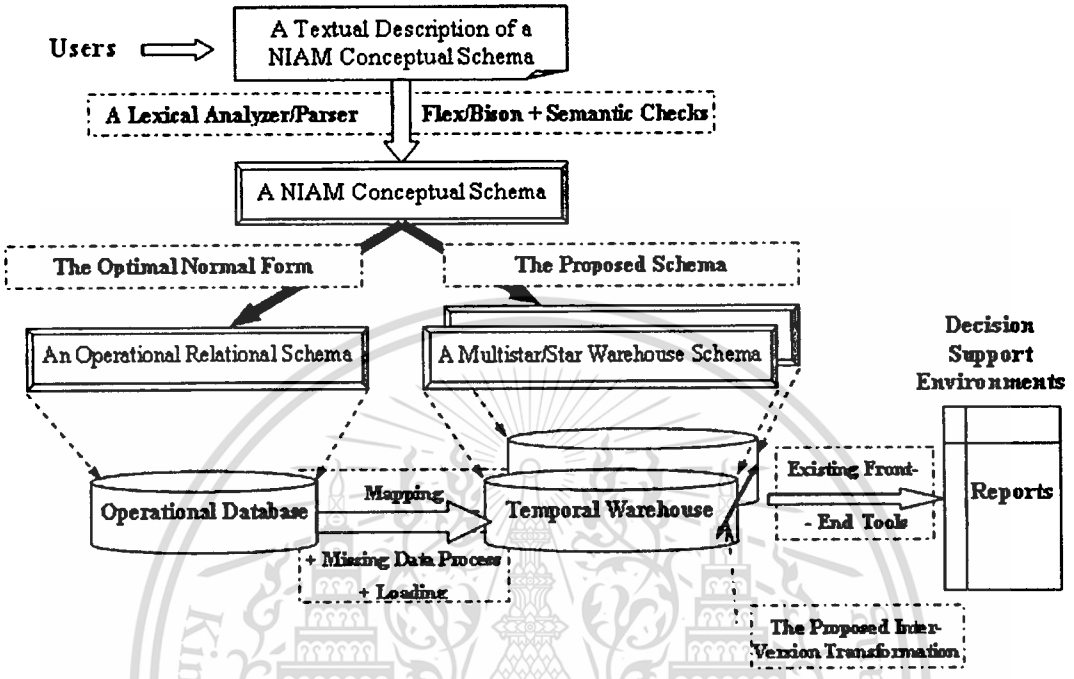


Figure 3.1 The Proposed Temporal Information Warehouse Architecture (TIWA1)

The three-tier TIWA1 architecture in Figure 3.1 shows considerations on how to achieve a temporal information warehouse through a few addressed tasks. All tasks that need to be resolved in the cycle are derived from the literature review in the previous chapter although some of them were identified for warehouse schema evolutions, for the multiple source integration, and so forth a long time ago [27]. These tasks will be tackled phase by phase along the flow of data.

From Figure 3.1, it is easy to realize that we took advantage of the NIAM/ORM modeling to construct a temporal information warehouse using the TIWA1 architecture.

Starting from a textual description which allows a NIAM/ORM conceptual schema to be described in a deep-structured natural language, an expected NIAM/ORM schema will be redefined in terms of NIAM/ORM elements after the execution of the lexical analyzer and the parser that are reached using the Flex/Bison tools plus semantic checks. Then, our proposed schema transformation rules are applied on the NIAM/ORM schema so that a warehouse schema is automatically and directly generated for the

structure of a temporal warehouse version. In addition to the proposed schema transformation from a NIAM/ORM conceptual schema to a warehouse schema, the Optimal Normal Form (ONF) algorithm in [18] (or the Relational Mapping Procedure in [52]) is employed so that metadata about operational sources are easily gathered from the mentioned NIAM/ORM conceptual schema. The metadata play a very important role to establish mappings between an operational source and a selected warehouse version. At this stage, operational data are asked to pass the process of missing data before the loading of operational data into their appropriate version is executed. Hence, other subtasks at the cleansing phase such as noisy data processes, data normalization, etc. are assumed to be completely satisfied. With the treatment of missing data at the back-end side and the proposed inter-version transformation technique for a temporal warehouse, warehouse data are significantly available for some existing front-end tools to make analysis reports.

Above all, the NIAM/ORM modeling is a means for user modeling of their warehouse and for the underlying implementation model of our system. This is due to the fact that the NIAM/ORM modeling allows a fact-oriented approach convenient for users to model their systems by using simple English sentences to express their requirements. Furthermost, relational tables can be obtained in the fifth normal form from a NIAM/ORM conceptual schema by carrying out the ONF algorithm so that the design of a large number of operational databases which have been well designed by using the NIAM/ORM modeling so far is made use of for the design of their related temporal warehouse.

In summary, we are going to further discuss the treatment of missing data at the back-end side and gathering metadata of operational sources from their NIAM/ORM conceptual schemas for the establishment of source-to-warehouse mappings, the transformation of a NIAM/ORM conceptual schema to a (multi) star warehouse schema, and the measure data transformation among warehouse versions.

Chapter 4

The Extraction, Transformation, and Loading Process

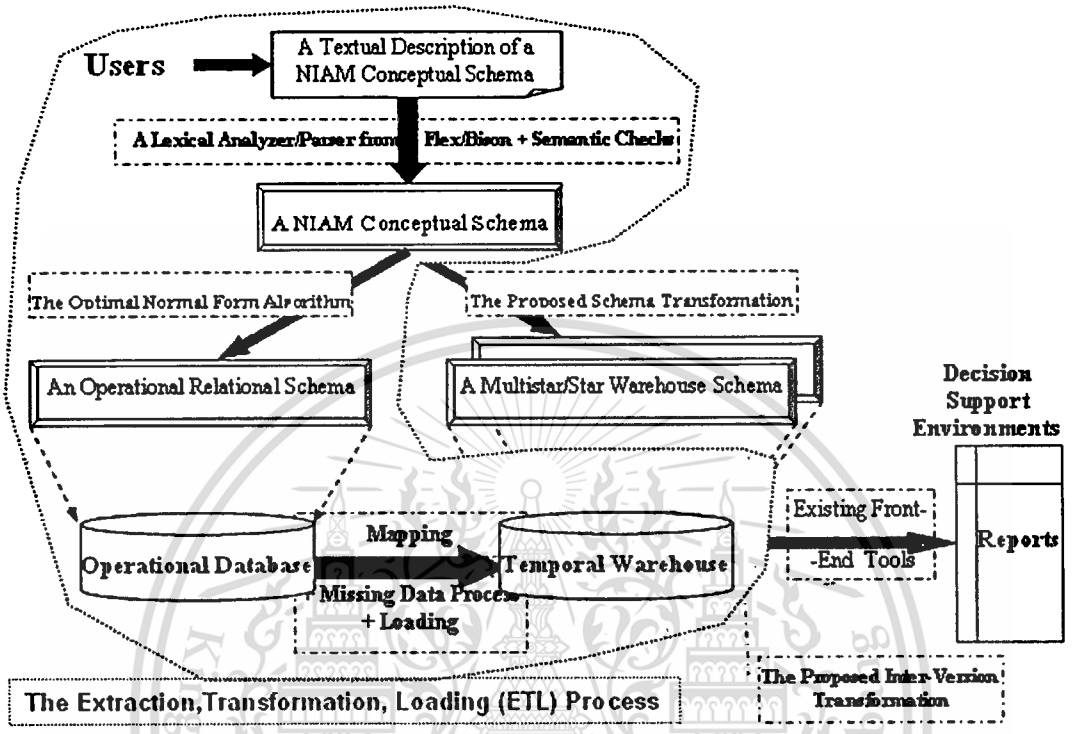


Figure 4.1 A Sub-Architecture of the ETL Process in a Temporal Information Warehouse System

This chapter will take into account the back-end side of a temporal information warehouse, particularly the extraction, transformation, loading (ETL) process. Up to now, there has been little suggestion on the ETL process in a temporal data warehouse system. In order to fulfill our temporal information warehouse building cycle, the ETL process is going to be considered. It is reduced to the ETL process in a non-temporal warehouse system where a warehouse now corresponds to a warehouse version of a temporal warehouse system. Be aware that a temporal information warehouse we are expecting is a warehouse with schema versioning as previously stated in the chapter 3.

As analyzed in [55], the ETL process is one of major tasks in the warehouse building cycle and makes up the cycle of more than 50% because many crucial subtasks included in the ETL process have a strong impact on the quality of warehouse data. These subtasks are transforming, integrating, normalizing, cleaning, and so forth

up to particular applications. Among them, handling missing measure data is one of the most interesting problems to be investigated in this research work.

On the other hand, metadata play a very important role in the warehouse building cycle as well as in the warehouse maintenance. The ETL process also requires metadata of operational sources to be managed for the establishment of mappings from operational databases to warehouses, and to warehouse versions in our context.

Hence, this chapter will discuss two sections, one for the gathering of source metadata and one for the handling of missing measure data as pointed out in Figure 4.1. Once source metadata together with warehouse metadata are available, it is trivial to construct relationships between source elements and warehouse elements for the loading task. The loading task will take place right after the missing data process at the cleansing phase is accomplished.

4.1 Gathering Source Metadata

The gathering of source metadata is carried out by manipulating a NIAM/ORM conceptual schema that describes an operational database at the conceptual level via a deep-structured natural language interface.

A NIAM/ORM context-free grammar for a NIAM/ORM conceptual schema specification is invented in Figure 5.4. The grammar is context-free in the Backus-Naur form (BNF) where uXXX are terminal symbols from user inputs, <xxx> are non-terminal symbols. XXX and xxx are reserved terminal symbols which are not allowed to be used as names of object types or roles. All reserved terminal symbols are case-sensitive. Abiding by the grammar, users input simple English sentences describing a NIAM/ORM conceptual schema. A lexical analyzer/parser gained by using the Flex/Bison tools will work on the specification to redefine it in terms of NIAM/ORM elements. All obtained information about the conceptual schema is saved in meta tables in the fifth normal form (5NF) produced from the NIAM/ORM conceptual meta schema in Figure 5.2. An elaborate explanation about the NIAM/ORM context-free grammar will be given in the chapter 5.

The NIAM/ORM conceptual meta schema is used to keep information of all NIAM/ORM elements and provide the Optimal Normal Form (ONF) algorithm [11] with

the information of a respective NIAM/ORM conceptual schema to achieve relational schemas of an operational source. By doing that, metadata of the source are fully collected so that users are able to establish mappings for the loading of operational data into a warehouse version. The achieved source metadata are stored along with information on source-to-warehouse mappings in 5NF meta tables also transformed from the back-end conceptual meta schema in Figure 4.2 by the ONF algorithm.

Given a NIAM/ORM conceptual schema described with simple English sentences using the NIAM/ORM context-free grammar in Figure 4.3 and visualized using Microsoft Visio for Enterprise Architect in Figure 4.5, source metadata are captured through a respective relational schema in Figure 4.4 that is generated from an SQL script in Data Definition Language (DDL) in Figure 4.6. This example is developed for the database FoodMart 2000 in the Microsoft SQL Server 2000 (Enterprise Edition).

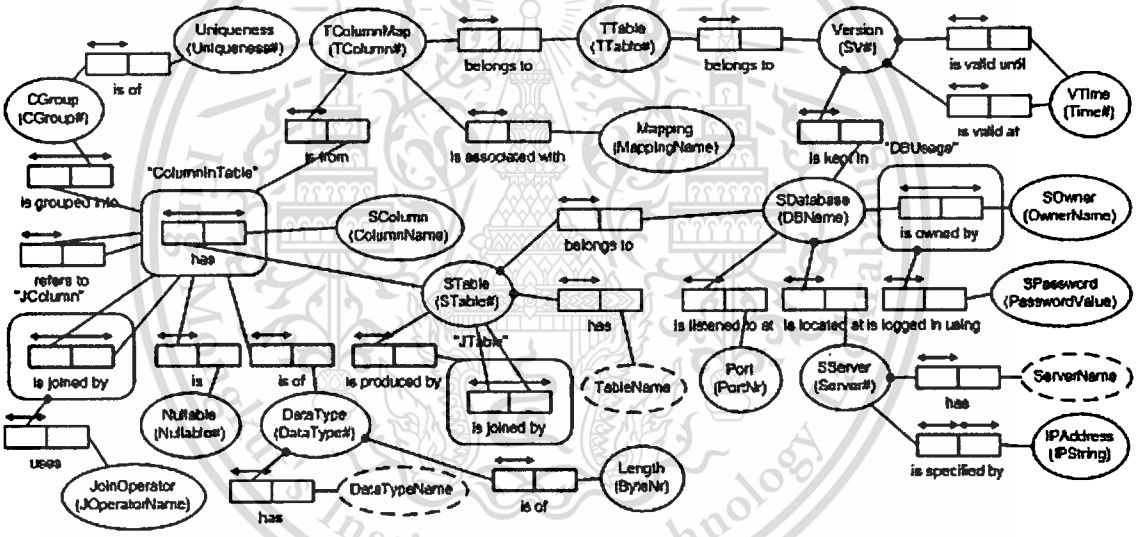


Figure 4.2 A Back-end Conceptual Meta Schema

Further from Figure 4.2, source-to-warehouse mappings are established at the column level. Column-to-column connections are explicitly specified by users (warehouse designers, warehouse administrators, and so on) for each warehouse version into which operational data are going to be transferred from a particular source known to users.

```
VERSION CSFoodMart10
        DESCRIPTION : "FoodMart2000".
        START VALIDITY : 01/01/1997.
        END VALIDITY : 12/31/9999.
END.
Account is_of/has AccountType.
```

Account has/describes Description.
City is_in/has District.
Customer opened_an_account_on/is_for Date.
Customer is_of/has Gender.
Customer has_an/is_paid_for IncomeAmount.
Customer is_at/is_with Location.
Customer is_with/is_for Marriage.
Customer has_a_first_name_of/belongs_to FirstName.
Customer has_a_last_name_of/belongs_to LastName.
Customer stays_at/is_for Place.
District belongs_to/has StateProvince.
Location is_in/has City.
Product is_of/has Brand.
Product belongs_to/has ProductClass.
Product has/belongs_to ProductName.
Product has/belongs_to RetailPrice.
Product has/belongs_to StockNumber.
ProductClass is_of/is ProductSubCategory.
ProductCategory belongs_to/has ProductDepartment.
ProductDepartment belongs_to/has ProductFamily.
ProductSubCategory is_classified_into/includes ProductCategory.
Promotion is_of/is_for Amount.
Promotion is_effective_on/is_for EffectiveDate.
Promotion is_ended_on/is_for EndedDate.
Promotion uses/is_used_by MediaType.
Promotion has/is_for PromotionName.
Region belongs_to/has Country.
StateProvince belongs_to/has Region.
Store is_located_at/has Location.
Store stays_at/belongs_to SPlace.
Store has/belongs_to StoreName.
Store is_managed_by/manages StoreManager.
Store is_of/has StoreType.
TimeByDay is_on/is Date.
TimeByDay is_on/is TheDay.
TimeByDay is_in/includes MonthOfYear.
TimeByDay is_in/includes WeekOfYear.
MonthOfYear is_the_th/is_for Month.
MonthOfYear is_in/includes QuarterOfYear.
MonthOfYear has/is_for TheMonth.
QuarterOfYear is_in/includes Year.
QuarterOfYear is_the_th/is_for Quarter.
WeekOfYear is_the_th/is_for Week.
WeekOfYear is_in/includes Year.
Warehouse is_at/is_with Location.
Warehouse is_owned_by/owns Owner.
Warehouse stays_at/has WPlace.
Warehouse belongs_to/includes WarehouseClass.
Warehouse has/belongs_to WarehouseName.
WarehouseClass has/describes WDescription.
Expense : Account of ExpenseAmount is_used_for Store on TimeByDay.
(Account Store TimeByDay) IS UNIQUE.
InventorySales : Warehouse serves Store for Product on TimeByDay with_the_sales_of InventoryAmount.
(Warehouse Store Product TimeByDay) IS UNIQUE.
Sales : Customer buys Product of_the SalesAmount with Promotion at Store on TimeByDay.
(Customer Product Promotion Store TimeByDay) IS UNIQUE.
CONSTRAINT
Account IS UNIQUELY IDENTIFIED BY AccountId : int(4).
AccountType IS UNIQUELY IDENTIFIED BY AccType : varchar(20).
Description IS UNIQUELY IDENTIFIED BY ADescription : nvarchar(50).
ExpenseAmount IS UNIQUELY IDENTIFIED BY AExpense : float(8).
Store IS UNIQUELY IDENTIFIED BY StoreId : int(4).
TimeByDay IS UNIQUELY IDENTIFIED BY TimeId : int(4).
City IS UNIQUELY IDENTIFIED BY CityName : nvarchar(50).
District IS UNIQUELY IDENTIFIED BY DistrictName : nvarchar(50).
Customer IS UNIQUELY IDENTIFIED BY CustomerId : int(4).
Date IS UNIQUELY IDENTIFIED BY TheDate : datetime(8).

Gender IS UNIQUELY IDENTIFIED BY GenderCode : nvarchar(1).
IncomeAmount IS UNIQUELY IDENTIFIED BY AIncome : money(8).
Location IS UNIQUELY IDENTIFIED BY LocationId : int(4).
Marriage IS UNIQUELY IDENTIFIED BY MaritalStatus : nvarchar(1).
Place IS UNIQUELY IDENTIFIED BY Address : nvarchar(100).
Product IS UNIQUELY IDENTIFIED BY ProductId : int(4).
SalesAmount IS UNIQUELY IDENTIFIED BY ASales : float(8).
Promotion IS UNIQUELY IDENTIFIED BY PromotionId : int(4).
Brand IS UNIQUELY IDENTIFIED BY BrandName : nvarchar(50).
ProductClass IS UNIQUELY IDENTIFIED BY ProductClassId : int(4).
RetailPrice IS UNIQUELY IDENTIFIED BY ARetailPrice : money(8).
StockNumber IS UNIQUELY IDENTIFIED BY AStockNumber : float(8).
ProductSubCategory IS UNIQUELY IDENTIFIED BY ProductSubCategoryId : nvarchar(15).
ProductCategory IS UNIQUELY IDENTIFIED BY ProductCategoryId : nvarchar(15).
ProductDepartment IS UNIQUELY IDENTIFIED BY ProductDepartmentId : nvarchar(15).
ProductFamily IS UNIQUELY IDENTIFIED BY ProductFamilyId : nvarchar(15).
Amount IS UNIQUELY IDENTIFIED BY AmountMoney : money(8).
EffectiveDate IS UNIQUELY IDENTIFIED BY EffDate : datetime(8).
EndedDate IS UNIQUELY IDENTIFIED BY EndDate : datetime(8).
MediaType IS UNIQUELY IDENTIFIED BY MediaTypeP : nvarchar(255).
Region IS UNIQUELY IDENTIFIED BY RegionName : nvarchar(50).
Country IS UNIQUELY IDENTIFIED BY CountryName : nvarchar(50).
StateProvince IS UNIQUELY IDENTIFIED BY StateName : nvarchar(50).
SPlace IS UNIQUELY IDENTIFIED BY SAddress : nvarchar(100).
StoreManager IS UNIQUELY IDENTIFIED BY SManagerName : nvarchar(255).
StoreType IS UNIQUELY IDENTIFIED BY SType : nvarchar(255).
MonthOfYear IS UNIQUELY IDENTIFIED BY YMonth : nvarchar(8).
Month IS UNIQUELY IDENTIFIED BY MonthNo: smallint(2).
QuarterOfYear IS UNIQUELY IDENTIFIED BY YQuarter : nvarchar(8).
Quarter IS UNIQUELY IDENTIFIED BY QuarterNo: nvarchar(2).
Year IS UNIQUELY IDENTIFIED BY TheYear : nvarchar(4).
WeekOfYear IS UNIQUELY IDENTIFIED BY YWeek: nvarchar(8).
Week IS UNIQUELY IDENTIFIED BY WeekNo: smallint(2).
Warehouse IS UNIQUELY IDENTIFIED BY WarehouseId : int(4).
Owner IS UNIQUELY IDENTIFIED BY OwnerName : nvarchar(50).
InventoryAmount IS UNIQUELY IDENTIFIED BY AInventory : float(8).
WPlace IS UNIQUELY IDENTIFIED BY WAddress : nvarchar(255).
WarehouseClass IS UNIQUELY IDENTIFIED BY WarehouseClassId : int(4).
WDescription IS UNIQUELY IDENTIFIED BY WHDescription : nvarchar(255).
EACH Account is_of EXACTLY ONE AccountType.
EACH AccountType has ZERO OR MORE Account.
EACH Account has ZERO OR ONE Description.
EACH Description describes ZERO OR MORE Account.
EACH City is_in EXACTLY ONE District.
EACH District has ONE OR MORE City.
EACH Customer opened_an_account_on ZERO OR ONE Date.
EACH Date is_for ZERO OR MORE Customer.
EACH Customer is_of EXACTLY ONE Gender.
EACH Gender has ONE OR MORE Customer.
EACH Customer has_an EXACTLY ONE IncomeAmount.
EACH IncomeAmount is_paid_for ZERO OR MORE Customer.
EACH Customer is_at EXACTLY ONE Location.
EACH Location is_with ZERO OR MORE Customer.
EACH Customer is_with EXACTLY ONE Marriage.
EACH Marriage is_for ZERO OR MORE Customer.
EACH Customer has_a_first_name_of EXACTLY ONE FirstName.
EACH FirstName belongs_to ZERO OR MORE Customer.
EACH Customer has_a_last_name_of EXACTLY ONE LastName.
EACH LastName belongs_to ZERO OR MORE Customer.
EACH Customer stays_at EXACTLY ONE Place.
EACH Place is_for ZERO OR MORE Customer.
EACH District belongs_to EXACTLY ONE StateProvince.
EACH StateProvince has ONE OR MORE District.
EACH Location is_in EXACTLY ONE City.
EACH City has ONE OR MORE Location.
EACH Product is_of EXACTLY ONE Brand.
EACH Brand has ZERO OR MORE Product.

EACH Product belongs_to EXACTLY ONE ProductClass.
EACH ProductClass has ZERO OR MORE Product.
EACH Product has EXACTLY ONE ProductName.
EACH ProductName belongs_to ZERO OR MORE Product.
EACH Product has EXACTLY ONE RetailPrice.
EACH RetailPrice belongs_to ZERO OR MORE Product.
EACH Product has EXACTLY ONE StockNumber.
EACH StockNumber belongs_to ZERO OR MORE Product.
EACH ProductClass is_of EXACTLY ONE ProductSubCategory.
EACH ProductSubCategory is ZERO OR MORE ProductClass.
EACH ProductCategory belongs_to EXACTLY ONE ProductDepartment.
EACH ProductDepartment has ZERO OR MORE ProductCategory.
EACH ProductDepartment belongs_to EXACTLY ONE ProductFamily.
EACH ProductFamily has ZERO OR MORE ProductDepartment.
EACH ProductSubCategory is_classified_into EXACTLY ONE ProductCategory.
EACH ProductCategory includes ZERO OR MORE ProductSubCategory.
EACH Promotion is_of EXACTLY ONE Amount.
EACH Amount is_for ZERO OR MORE Promotion.
EACH Promotion is_effective_on EXACTLY ONE EffectiveDate.
EACH EffectiveDate is_for ZERO OR MORE Promotion.
EACH Promotion is_ended_on EXACTLY ONE EndedDate.
EACH EndedDate is_for ZERO OR MORE Promotion.
EACH Promotion uses ZERO OR ONE MediaType.
EACH MediaType is_used_by ZERO OR MORE Promotion.
EACH Promotion has EXACTLY ONE PromotionName.
EACH PromotionName is_for ZERO OR MORE Promotion.
EACH Region belongs_to EXACTLY ONE Country.
EACH Country has ONE OR MORE Region.
EACH StateProvince belongs_to EXACTLY ONE Region.
EACH Region has ONE OR MORE StateProvince.
EACH Store is_located_at EXACTLY ONE Location.
EACH Location has ZERO OR MORE Store.
EACH Store stays_at EXACTLY ONE SPlace.
EACH SPlace belongs_to ZERO OR MORE Store.
EACH Store has EXACTLY ONE StoreName.
EACH StoreName belongs_to ZERO OR MORE Store.
EACH Store is_managed_by EXACTLY ONE StoreManager.
EACH StoreManager manages ZERO OR MORE Store.
EACH Store is_of EXACTLY ONE StoreType.
EACH StoreType has ZERO OR MORE Store.
EACH TimeByDay is_on EXACTLY ONE Date.
EACH Date is ONE OR MORE TimeByDay.
EACH TimeByDay is_on EXACTLY ONE TheDay.
EACH TheDay is ONE OR MORE TimeByDay.
EACH TimeByDay is_in EXACTLY ONE MonthOfYear.
EACH MonthOfYear includes ONE OR MORE TimeByDay.
EACH MonthOfYear is_the_th EXACTLY ONE Month.
EACH Month is_for ONE OR MORE MonthOfYear.
EACH MonthOfYear is_in EXACTLY ONE QuarterOfYear.
EACH QuarterOfYear includes ONE OR MORE MonthOfYear.
EACH MonthOfYear has EXACTLY ONE TheMonth.
EACH TheMonth is_for ONE OR MORE MonthOfYear.
EACH QuarterOfYear is_in EXACTLY ONE Year.
EACH Year includes ONE OR MORE QuarterOfYear.
EACH QuarterOfYear is_the_th EXACTLY ONE Quarter.
EACH Quarter is_for ONE OR MORE QuarterOfYear.
EACH TimeByDay is_in EXACTLY ONE WeekOfYear.
EACH WeekOfYear includes ONE OR MORE TimeByDay.
EACH WeekOfYear is_the_th EXACTLY ONE Week.
EACH Week is_for ONE OR MORE WeekOfYear.
EACH WeekOfYear is_in EXACTLY ONE Year.
EACH Year includes ONE OR MORE WeekOfYear.
EACH Warehouse is_at EXACTLY ONE Location.
EACH Location is_with ZERO OR MORE Warehouse.
EACH Warehouse is_owned_by EXACTLY ONE Owner.
EACH Owner owns ZERO OR MORE Warehouse.
EACH Warehouse stays_at EXACTLY ONE WPlace.

```
        EACH WPlace has  ZERO OR MORE Warehouse.
        EACH Warehouse belongs_to EXACTLY ONE WarehouseClass.
        EACH WarehouseClass includes  ZERO OR MORE Warehouse.
        EACH Warehouse has EXACTLY ONE WarehouseName.
        EACH WarehouseName belongs_to  ZERO OR MORE Warehouse.
        EACH WarehouseClass has EXACTLY ONE WDescription.
        EACH WDescription describes  ZERO OR MORE WarehouseClass.
END.
DATATYPE
        FirstName : nvarchar(100).
        LastName : nvarchar(100).
        ProductName : nvarchar(255).
        PromotionName : nvarchar(100).
        StoreName : nvarchar(255).
        TheDay: nvarchar(15).
        TheMonth: nvarchar(15).
        WarehouseName : nvarchar(100).
END.
```

Figure 4.3 The Textual Description of the NIAM Conceptual Schema Modeling FoodMart 2000

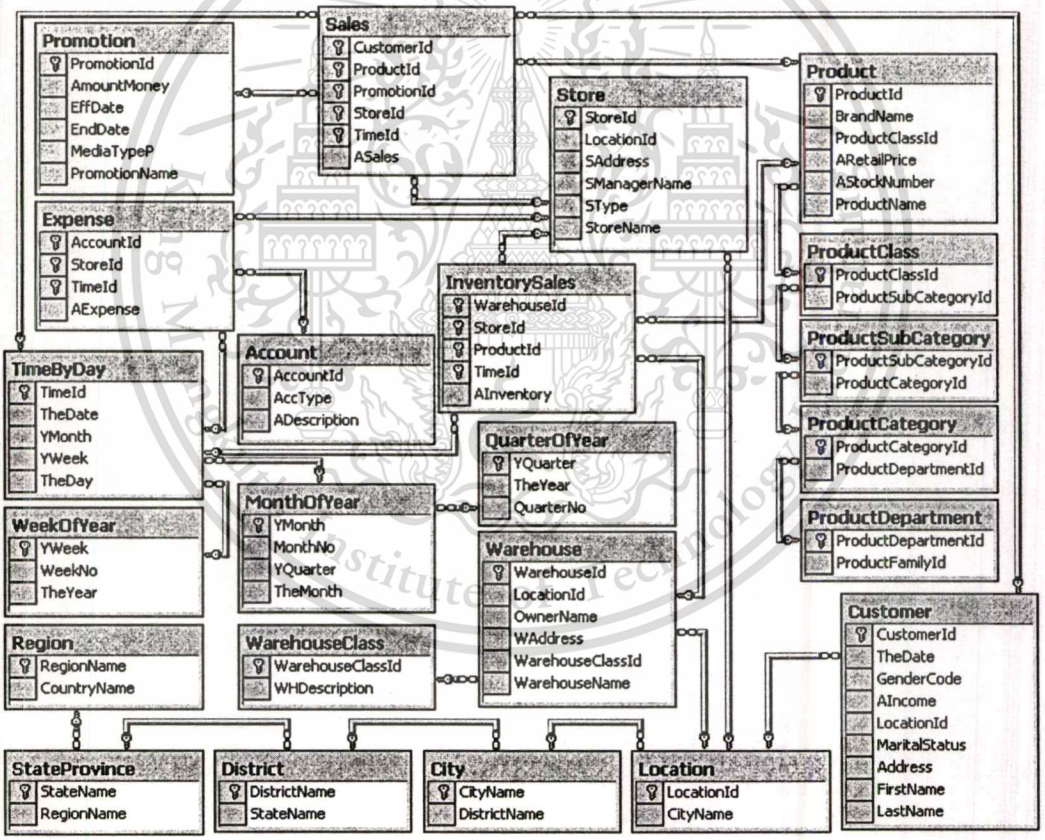


Figure 4.4 A Respective Relational Schema for the FoodMart 2000 Example

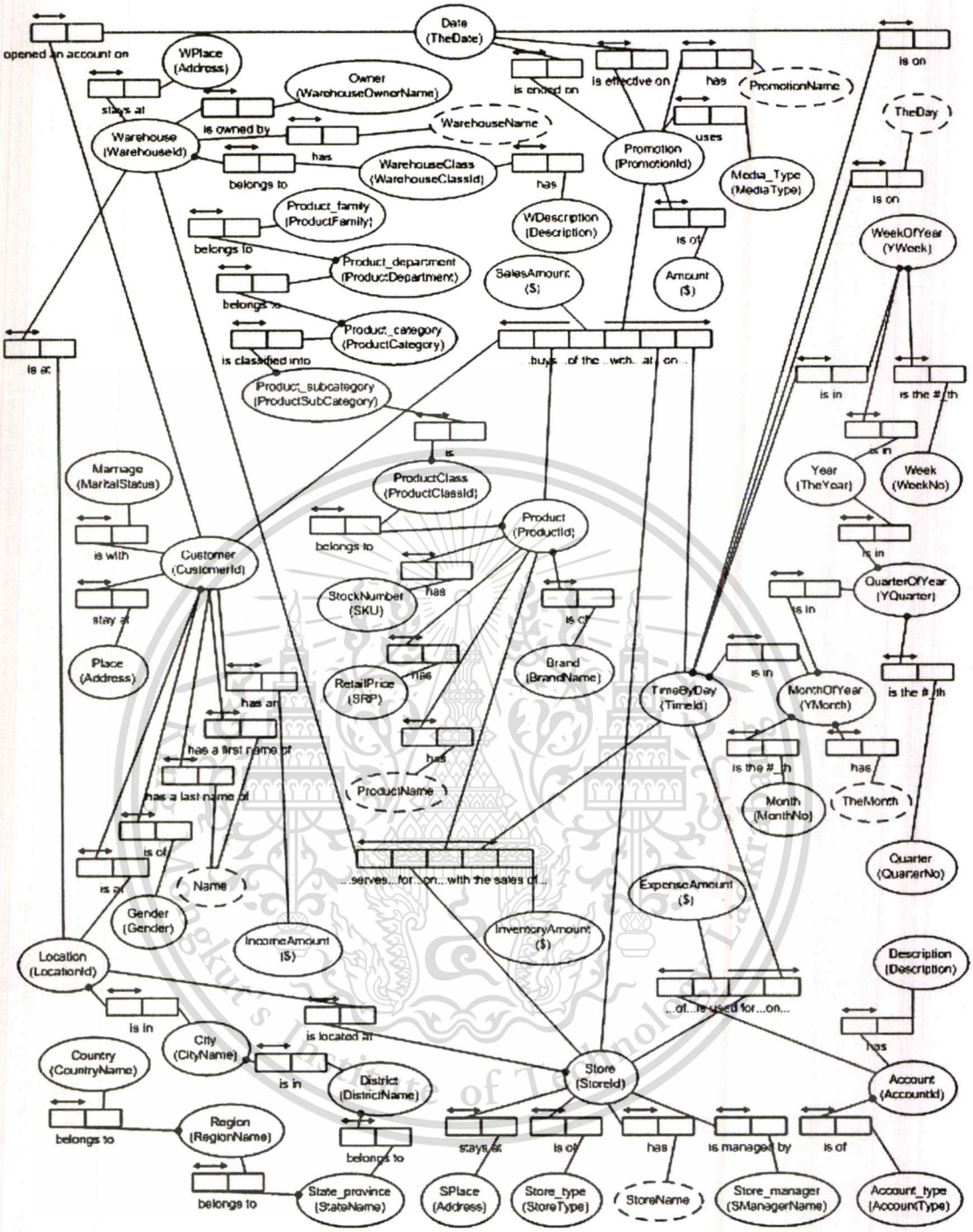


Figure 4.5 The Visualized NIAM Conceptual Schema Modeling the FoodMart 2000 Example

```
CREATE TABLE Expense
(
AccountId int,
StoreId int,
TimeId int,
AExpense float,
CONSTRAINT Expense_Key PRIMARY KEY (AccountId, StoreId, TimeId)
```

```
);
CREATE TABLE InventorySales
(
WarehouseId int,
StoreId int,
ProductId int,
TimeId int,
AInventory float,
CONSTRAINT InventorySales_Key PRIMARY KEY (WarehouseId, StoreId, ProductId, TimeId)
);
CREATE TABLE Sales
(
CustomerId int,
ProductId int,
PromotionId int,
StoreId int,
TimeId int,
ASales float,
CONSTRAINT Sales_Key PRIMARY KEY (CustomerId, ProductId, PromotionId, StoreId, TimeId)
);
CREATE TABLE WarehouseClass
(
WarehouseClassId int,
WHDescription nvarchar(255),
CONSTRAINT WarehouseClass_Key PRIMARY KEY (WarehouseClassId)
);
CREATE TABLE Warehouse
(
WarehouseId int,
LocationId int,
OwnerName nvarchar(50),
WAddress nvarchar(255),
WarehouseClassId int,
WarehouseName nvarchar(100),
CONSTRAINT Warehouse_Key PRIMARY KEY (WarehouseId)
);
CREATE TABLE WeekOfYear
(
YWeek nvarchar(8),
WeekNo smallint,
TheYear nvarchar(4),
CONSTRAINT WeekOfYear_Key PRIMARY KEY (YWeek)
);
CREATE TABLE QuarterOfYear
(
YQuarter nvarchar(8),
TheYear nvarchar(4),
QuarterNo nvarchar(2),
CONSTRAINT QuarterOfYear_Key PRIMARY KEY (YQuarter)
);
CREATE TABLE MonthOfYear
(
YMonth nvarchar(8),
MonthNo smallint,
YQuarter nvarchar(8),
TheMonth nvarchar(15),
CONSTRAINT MonthOfYear_Key PRIMARY KEY (YMonth)
);
CREATE TABLE StateProvince
(
```

```
StateName nvarchar(50),
RegionName nvarchar(50),
CONSTRAINT StateProvince_Key PRIMARY KEY (StateName)
);
CREATE TABLE Region
(
RegionName nvarchar(50),
CountryName nvarchar(50),
CONSTRAINT Region_Key PRIMARY KEY (RegionName)
);
CREATE TABLE ProductDepartment
(
ProductDepartmentId nvarchar(15),
ProductFamilyId nvarchar(15),
CONSTRAINT ProductDepartment_Key PRIMARY KEY (ProductDepartmentId)
);
CREATE TABLE ProductCategory
(
ProductCategoryId nvarchar(15),
ProductDepartmentId nvarchar(15),
CONSTRAINT ProductCategory_Key PRIMARY KEY (ProductCategoryId)
);
CREATE TABLE ProductSubCategory
(
ProductSubCategoryId nvarchar(15),
ProductCategoryId nvarchar(15),
CONSTRAINT ProductSubCategory_Key PRIMARY KEY (ProductSubCategoryId)
);
CREATE TABLE ProductClass
(
ProductClassId int,
ProductSubCategoryId nvarchar(15),
CONSTRAINT ProductClass_Key PRIMARY KEY (ProductClassId)
);
CREATE TABLE Promotion
(
PromotionId int,
AmountMoney money,
EffDate datetime,
EndDate datetime,
MediaTypeP nvarchar(255),
PromotionName nvarchar(100),
CONSTRAINT Promotion_Key PRIMARY KEY (PromotionId)
);
CREATE TABLE Product
(
ProductId int,
BrandName nvarchar(50),
ProductClassId int,
ARetailPrice money,
AStockNumber float,
ProductName nvarchar(255),
CONSTRAINT Product_Key PRIMARY KEY (ProductId)
);
CREATE TABLE Location
(
LocationId int,
CityName nvarchar(50),
CONSTRAINT Location_Key PRIMARY KEY (LocationId)
);
```

```
CREATE TABLE Customer
(
CustomerId int,
TheDate datetime,
GenderCode nvarchar(1),
AIncome money,
LocationId int,
MaritalStatus nvarchar(1),
Address nvarchar(100),
FirstName nvarchar(100),
LastName nvarchar(100),
CONSTRAINT Customer_Key PRIMARY KEY (CustomerId)
);
CREATE TABLE District
(
DistrictName nvarchar(50),
StateName nvarchar(50),
CONSTRAINT District_Key PRIMARY KEY (DistrictName)
);
CREATE TABLE City
(
CityName nvarchar(50),
DistrictName nvarchar(50),
CONSTRAINT City_Key PRIMARY KEY (CityName)
);
CREATE TABLE TimeByDay
(
TimeId int,
TheDate datetime,
YMonth nvarchar(8),
YWeek nvarchar(8),
TheDay nvarchar(15),
CONSTRAINT TimeByDay_Key PRIMARY KEY (TimeId)
);
CREATE TABLE Store
(
StoreId int,
LocationId int,
SAddress nvarchar(100),
SManagerName nvarchar(255),
SType nvarchar(255),
StoreName nvarchar(255),
CONSTRAINT Store_Key PRIMARY KEY (StoreId)
);
CREATE TABLE Account
(
AccountId int,
AccType varchar(20),
ADescription nvarchar(50),
CONSTRAINT Account_Key PRIMARY KEY (AccountId)
);
ALTER TABLE Expense ADD CONSTRAINT Expense_AccountId_FK FOREIGN KEY(AccountId)
REFERENCES Account(AccountId);
ALTER TABLE Expense ADD CONSTRAINT Expense_StoreId_FK FOREIGN KEY(StoreId)
REFERENCES Store(StoreId);
ALTER TABLE Expense ADD CONSTRAINT Expense_TimeId_FK FOREIGN KEY(TimeId)
REFERENCES TimeByDay(TimeId);
ALTER TABLE InventorySales ADD CONSTRAINT InventorySales_WarehouseId_FK FOREIGN
KEY(WarehouseId) REFERENCES Warehouse(WarehouseId);
```

```
ALTER TABLE InventorySales ADD CONSTRAINT InventorySales_StoreId_FK FOREIGN
KEY(StoreId) REFERENCES Store(StoreId);
ALTER TABLE InventorySales ADD CONSTRAINT InventorySales_ProductId_FK FOREIGN
KEY(ProductId) REFERENCES Product(ProductId);
ALTER TABLE InventorySales ADD CONSTRAINT InventorySales_TimeId_FK FOREIGN
KEY(TimeId) REFERENCES TimeByDay(TimeId);
ALTER TABLE Sales ADD CONSTRAINT Sales_CustomerId_FK FOREIGN KEY(CustomerId)
REFERENCES Customer(CustomerId);
ALTER TABLE Sales ADD CONSTRAINT Sales_ProductId_FK FOREIGN KEY(ProductId)
REFERENCES Product(ProductId);
ALTER TABLE Sales ADD CONSTRAINT Sales_PromotionId_FK FOREIGN KEY(PromotionId)
REFERENCES Promotion(PromotionId);
ALTER TABLE Sales ADD CONSTRAINT Sales_StoreId_FK FOREIGN KEY(StoreId)
REFERENCES Store(StoreId);
ALTER TABLE Sales ADD CONSTRAINT Sales_TimeId_FK FOREIGN KEY(TimeId)
REFERENCES TimeByDay(TimeId);
ALTER TABLE Warehouse ADD CONSTRAINT Warehouse_LocationId_FK FOREIGN
KEY(LocationId) REFERENCES Location(LocationId);
ALTER TABLE Warehouse ADD CONSTRAINT Warehouse_WarehouseClassId_FK FOREIGN
KEY(WarehouseClassId) REFERENCES WarehouseClass(WarehouseClassId);
ALTER TABLE MonthOfYear ADD CONSTRAINT MonthOfYear_YQuarter_FK FOREIGN
KEY(YQuarter) REFERENCES QuarterOfYear(YQuarter);
ALTER TABLE StateProvince ADD CONSTRAINT StateProvince_RegionName_FK FOREIGN
KEY(RegionName) REFERENCES Region(RegionName);
ALTER TABLE ProductCategory ADD CONSTRAINT ProductCategory_ProductDepartmentId_FK
FOREIGN KEY(ProductDepartmentId) REFERENCES ProductDepartment(ProductDepartmentId);
ALTER TABLE ProductSubCategory ADD CONSTRAINT
ProductSubCategory_ProductCategoryId_FK FOREIGN KEY(ProductCategoryId) REFERENCES
ProductCategory(ProductCategoryId);
ALTER TABLE ProductClass ADD CONSTRAINT ProductClass_ProductSubCategoryId_FK
FOREIGN KEY(ProductSubCategoryId) REFERENCES
ProductSubCategory(ProductSubCategoryId);
ALTER TABLE Product ADD CONSTRAINT Product_ProductClassId_FK FOREIGN
KEY(ProductClassId) REFERENCES ProductClass(ProductClassId);
ALTER TABLE Location ADD CONSTRAINT Location_CityName_FK FOREIGN KEY(CityName)
REFERENCES City(CityName);
ALTER TABLE Customer ADD CONSTRAINT Customer_LocationId_FK FOREIGN
KEY(LocationId) REFERENCES Location(LocationId);
ALTER TABLE District ADD CONSTRAINT District_StateName_FK FOREIGN KEY(StateName)
REFERENCES StateProvince(StateName);
ALTER TABLE City ADD CONSTRAINT City_DistrictName_FK FOREIGN KEY(DistrictName)
REFERENCES District(DistrictName);
ALTER TABLE TimeByDay ADD CONSTRAINT TimeByDay_YMonth_FK FOREIGN
KEY(YMonth) REFERENCES MonthOfYear(YMonth);
ALTER TABLE TimeByDay ADD CONSTRAINT TimeByDay_YWeek_FK FOREIGN
KEY(YWeek) REFERENCES WeekOfYear(YWeek);
ALTER TABLE Store ADD CONSTRAINT Store_LocationId_FK FOREIGN KEY(LocationId)
REFERENCES Location(LocationId);
```

Figure 4.6 An SQL Script in DDL for an Operational Database corresponding to Figures 4.3 and 4.4

4.2 Handling Missing Measure Data in Data Warehousing

In data warehousing, the handling of missing data is a subtask of the ETL (Extraction, Transformation and Loading) process. It is important to have complete data

in a data warehouse in order to produce accurate reports for the management. However, missing data are common in practice and the need to have close estimations is very clear. This section focuses on the estimation of missing values for data warehouses. We employ the popular K-means clustering technique with the combination of the probabilistic theory to obtain probabilities from sample measure data, individual probabilities for some specified period of time and average probabilities for many periods of time from average measure data which are computed using the K-means clustering technique. Each cell in a fact table with a missing value is filled in by a calculated value from its associated proportion and available values of members at the same level that are related to the same member at the higher level. After an evaluation using the Euclidean distance, it is found that our technique is more applicable and produces better estimated data than existing techniques.

4.2.1 Taxonomy of Missing Warehouse Data

For this taxonomy, two important factors are necessary to be considered. They are the amount of missing data and missing data mechanisms due to their crucial impacts on missing data treatments. Hence, it is worth investigating the various patterns of incomplete data.

A general taxonomy of missing data which may emerge is given as follows according to the relationship (dependency) between the missing data and their missing values and completely observed data [16]. Let Y denote a data vector composed of two parts, the first part is completely observed and the second one is potentially missing. In other words, Y = (Y observed, Y missing).

- Missing at Random (MAR): If the proportion of Y missing does not depend on missing values themselves, but may depend on values of Y observed, then Y missing are said to be missing at random. MAR implies that participants with incomplete data may differ from participants with complete data but the missing patterns are predictable from other complete patterns. When data are MAR, some powerful statistical techniques are available to deal with the missing data problem.
- Missing Completely at Random (MCAR): If the proportion of Y missing depends on neither Y observed nor missing values themselves that could have been collected or

recorded, then Y missing are said to be MCAR. However, MCAR is a very stringent assumption. It is probably rarely encountered in the real world of research.

- Non-ignorable missing (not missing at random – NMAR): the missing data mechanism is said to be ignorable if the data are MAR and the parameters that govern the missing data process are unrelated to the parameters to be estimated. Ignorability basically means that there is no need to model the missing data mechanism as part of the estimation process. In other words, the missing data are non-ignorable if the proportion of missing data depends on the missing values themselves. The usual treatment applied on NMAR data is deletion with the acceptably small amount of the missing data.

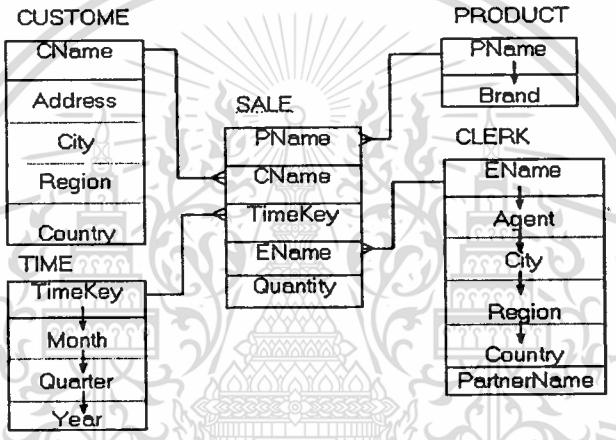


Figure 4.7 A Simple Warehouse Schema

From the taxonomy above, missing warehouse data are categorized according to the phenomena of missing data occurrences, the nature/the cause of missing data, the volume of missing data, and the kind of data in a target warehouse version into which missing data are going to be put. For a clear illustration, consider a simple star schema of a warehouse structure version SVi valid in a time interval [tsi, tei] where tei is greater than tsi with the dimension Time for normally tracking measure data on some periodic basis as presented in Figure 4.7.

4.2.1.1 Case 1

- Phenomenon: The data are not available at the extracting time, but they exist in the real world. For example, the field Address of the dimension Customer is nullable. Some row in the dimension CUSTOMER may occur without any value for Address. It is due to corresponding operational sources do not request users to provide a value

for this field. Hence, this field is sometimes left as NULL although the value for it exists in reality.

- Missing kind: Missing Completely at Random
- Volume: it is up to particular applications
- Type of respective warehouse data: Since the values of missing data are gathered from the users at the operational sources, the data into which missing values are going to be loaded are usually dimension data used for description.
- Solution: wait for the data at the next loading times to replace the old incomplete one

4.2.1.2 Case 2

- Phenomenon: The values of missing data don't exist in the real world at the extracting time but it will occur in the future. For example, the value for the field PartnerName of the dimension CLERK is not available at the extracting time T1 because he/she is independently working. After a little, he/she will cooperate with someone that is called her/his partner. Hence, at the extracting time T2 (T2 > T1), the value for that field will be different from NULL.
- Missing kind: Missing Completely at Random
- Volume: it is also unpredictable because it is due to real world situations.
- Type of respective warehouse data: dimension data for description
- Solution: wait for the data at the next loading times to replace the old incomplete one.

4.2.1.3 Case 3

- Phenomenon: The data are unavailable because there are no their existences. For example, there is a Customer C, a Product P, and a Clerk Cl. In January associated with the TimeKey T1, the customer C does not make any transaction for the product P with the clerk Cl. Hence, there is no value for Quantity corresponding to the tuple (C, P, Cl, T1) in the fact table SALE.
- Missing kind: Missing Completely at Random
- Volume: The volume of missing data is often high because fact tables are normally sparse in practice.
- Type of respective warehouse data: numeric measure data

- Solution: They are filled in with zeros or some known constant up to the nature of business transactions.

4.2.1.4 Case 4

- Phenomenon: the required data are unavailable because they are unable to be recorded in the operational sources. However, they are able to be derived from other complete data in the operational sources. For example, the detailed data are missing, but the summarized data at the higher levels are successfully collected. Another situation is that a measure value corresponding to one surrogate key from a dimension is missing, but the measure data for this combination of it with other surrogate keys of the dimension are successfully observed.
- Missing kind: Missing at Random
- Volume: It is unpredictable, but sometimes high.
- Type of respective warehouse data: numeric measure data
- Solution: techniques in [28] and [39], our proposed missing measure data handling technique

4.2.1.5 Case 5

- Phenomenon: The required data are not available because the operational system is unable to capture it during the transaction in which the data are yielded as usual. Consider the example in the case 3, but there existed the transaction of the customer C for the product P with the clerk Cl in January with the TimeKey T1. The value of Quantity for the tuple (C, P, Cl, T1) was unsuccessfully caught.
- Missing kind: Missing Completely at Random
- Volume: It is unpredictable due to actual operational systems.
- Type of respective warehouse data: numeric measure data
- Solution: techniques in [28] and [39], our proposed missing measure data handling technique

In summary, we will deal with the missing data problem in the cases 4 and 5. Other cases require more efforts. Anyway, there has been no solution for these cases up to now. Except for the cases 4 and 5, the treatment of missing warehouse data in other cases is still promising. In addition, this classification on missing warehouse data may miss out some cases in reality.

4.2.2 The Existing Missing Measure Data Handling Techniques

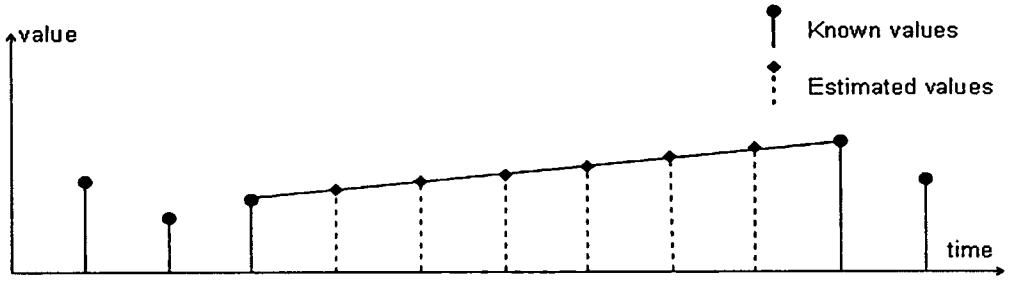


Figure 4.8 The Linear Regression used in [39]

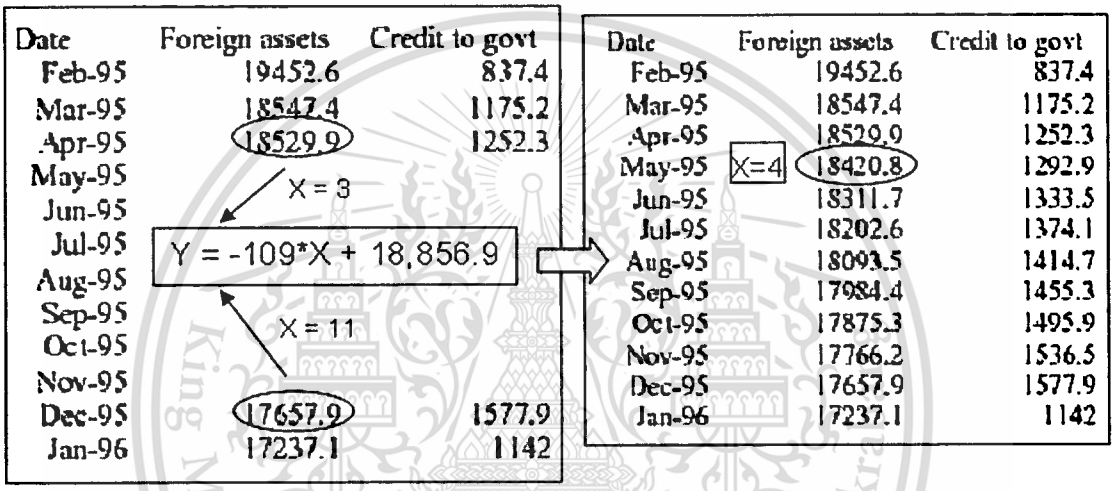


Figure 4.9 An Illustration on Filling in Missing Data in [39]

As early reviewed, two existing missing data handling techniques considered for the comparison are the linear regression in [39] and the use of average values in [28].

The technique with the linear regression in [39] proposes a clean view approach to the treatment of missing data by creating a view/sight of the original data after the data selection and the transformation with the cleaning rules such as the rule about the substitution of a value, and omitting a record. These rules will redefine data as time series data and how to apply the linear regression with two prior known values to fill in missing data as illustrated in Figure 4.9. Their solution will face difficulties with missing data which occur at the first and at the last tuple in time series data because at that time there is lack of information to compute parameters for the linear regression. Besides, using the linear regression implicitly assumes that data distributions abide by some

equations of straight lines in Figure 4.8.

Filling in missing data after the missing data detection with NULL values, the technique in [28] used the average value of the attribute values of all tuples associated with the same group members of which are similar in some interesting aspect. This approach to dealing with missing data might provide better estimated data than using only one average value to replace NULL values which occur in all tuples of a fact table in a data warehouse. However, it requires either grouping or clustering as shown in Figure 4.10. Their example in Table 41 and Table 42 uses grouping to assign the average of the attribute values of a class to a missing value. The average values are stored in a temporary relation tmp(Class, X).

```
insert into tmp (
        select Class, avg(X) as X
        from Measurements group by Class);
update Measurements
        set X = (select X from tmp
                where tmp.Class = Measurements.Class)
where X is null;
```

Figure 4.10 Filling in Missing Data in [28] using Average Values

Table 4.1 SALE with random missing data

PName	CName	TimeKey	EName	Quantity
P1	C1	T1	E1	100
P1	C2	T1	E1	150
P1	C2	T2	E1	<null>
P2	C2	T2	E1	200

Table 4.2 SALE after filling in missing data with the average value that is calculated for the group including only the customer C2

PName	CName	TimeKey	EName	Quantity
P1	C1	T1	E1	100
P1	C2	T1	E1	150
P1	C2	T2	E1	175
P2	C2	T2	E1	200

Above all, there is an implicit uniformity assumption on data distribution in their solution.

4.2.3 The Proposed Missing Measure Data Handling Technique

For both data warehouses and temporal data warehouses, by using probabilistics and the K-means clustering technique, estimated data are computed for the fields data

of which are missing after all other subtasks in the ETL process are well satisfied. The main idea behind our technique is obtaining probabilities from sample measure data, individual probabilities for some specified period of time and average probabilities for many periods of time from average measure data calculated using the K-means clustering technique. Each cell in fact tables which is value-missing is filled in from its respective proportion and known values of members at the same level that belong to the same higher level.

Firstly, users are asked to specify which fact table is target of the treatment of missing data and which dimension table associated with the fact table is selected as base of users' interest about time, a geographic concern, and so on.

Secondly, users are asked to specify the scope respectively, e.g., year for time, country for a geographic concern, or any higher dimension level than the key level for that chosen base dimension. This higher dimension level is called a base dimension level.

As previously stated, we propose two approaches to the calculation of probabilities corresponding to each member of the key in the base dimension table. One is obtaining individual probabilities for some specified period of time. The other is to obtain average probabilities for many periods of time.

For the first approach, let P_j for $j = \overline{1,n_t}$ be the probabilities for n_t members of the key level that belong to the same member of the selected higher base dimension level in the base dimension. Each P_j is below calculated:

$$P_j = \frac{Measure\_Data\_DM_{1j}}{\sum_{z=1}^{n_t} Measure\_Data\_DM_{1z}}, j = \overline{1,n_t} \qquad (4\text{-}1)$$

Where $Measure\_Data\_DM_{1j}$ for $j = \overline{1,n_t}$ in the measure data sample are input by users.

As for the second approach, let $\overline{P_j}$, where $j = \overline{1,n_t}$ be the probabilities for average measure data related to each dimension member of the key. These $\overline{P_j}$, where $j = \overline{1,n_t}$ are computed as follows from average measure data of each dimension member DM_{1j} where $j = \overline{1,n_t}$ of the key:

$$\overline{P_j} = \frac{\overline{Measure\_Data\_DM_{1j}}}{\sum_{z=1}^{n_t} \overline{Measure\_Data\_DM_{1z}}}, j = \overline{1,n_t} \qquad (4\text{-}2)$$

Where $\overline{Measure\_Data\_DM_{1j}}$ for j = $\overline{1, n_t}$ are the average measure data for each dimension member DM_{1j} for j =$\overline{1, n_t}$. It will be calculated using the K-means clustering technique. How to apply the K-means clustering technique for the achievement of $\overline{Measure\_Data\_DM_{1j}}$ for j = $\overline{1, n_t}$ will be minutely explained in the chapter 6.

Before going into details on how to fill in missing measure data in a fact table, we will set up a case study as follows.

Consider a fact table F associated with three dimensions Df, D1, and D2 where Df is the selected base dimension. Let the key level of Df be K, the key levels of D1 and D2 be K1 and K2 respectively. Let DL be the selected base dimension level in Df. Member relationships between members of K and members of DL are given in Table 4.3.

For the first approach with a single-period measure data sample, individual probabilities P(ki) for members ki of K are calculated from Table 4.3.

Table 4.3 A Single-period Measure Data Sample

DL	K	Measure
dl1	k1	m1
dl1	k2	m2
dl1	k3	m3
dl2	k4	m4
dl2	k5	m5
dl3	k6	m6
dl3	k7	m7
dl3	k8	m8
dl3	k9	m9
dl4	k10	m10
dl4	k11	m11

$$P(k1) = \frac{m1}{m1+m2+m3};$$

$$P(k2) = \frac{m2}{m1+m2+m3};$$

$$P(k3) = \frac{m3}{m1+m2+m3};$$

$$P(k4) = \frac{m4}{m4+m5};$$

..... $$P(k11) = \frac{m11}{m10+m11};$$

For the second approach with a multiple-period measure data sample, average probabilities $\overline{P}$(ki) for members of K are calculated from Table 4.4.

After accomplishing probabilities for all members of the key level in the base dimension in Table 4.5, the process of computing estimated values for missing measure data in the fact F is below performed in Table 4.6 where the nature of missing measure data is missing at random (MAR).

$$<null1 >= \frac{p1*m2}{p2}; \quad <null5 >= \frac{p5*m4}{p4}; \quad <null6 >= 0;$$

$$<null7 >= 0; \quad <null8 >= 0; \quad <null12 >= 0;$$

Table 4.4 A Multiple-period Measure Data Sample

DL	K	Measure
dl1	k1	m1,1
dl1	k2	m2,1
dl1	k3	m3,1
dl1	k1	m1,2
dl1	k2	m2,2
dl1	k3	m3,2
dl1	k1	m1,3
dl1	k2	m2,3
dl1	k3	m3,3
dl1	k1	m1,4
dl1	k2	m2,4
dl1	k3	m3,4
...	...	...
dl1	k3	m3,N
dl2	k4	m4,1
...	...	...

$$\bar{P}(k1) = \frac{\overline{m1}}{\overline{m1}+\overline{m2}+\overline{m3}}$$

$$\bar{P}(k2) = \frac{\overline{m2}}{\overline{m1}+\overline{m2}+\overline{m3}}$$

$$\bar{P}(k3) = \frac{\overline{m3}}{\overline{m1}+\overline{m2}+\overline{m3}}$$

$$\bar{P}(k4) = \frac{\overline{m4}}{\overline{m4}+\overline{m5}}$$

.
.
.

Where $\overline{mi}$ is the average measure data obtained from the K-means clustering technique

Table 4.5 Obtained Probabilities

DL	K	Proportion
dl1	k1	p1
dl1	k2	p2
dl1	k3	p3
dl2	k4	p4
dl2	k5	p5
dl1	k1	p1
dl1	k2	p2
dl1	k3	p3
dl1	k1	p1
dl1	k2	p2
dl1	k3	p3
dl2	k4	p4
dl2	k4	p4
dl2	k5	p5
...	....	....

Table 4.6 Randomly Missing Data in F

K	K1	K2	Measure
k1	k1,1	k2,1	<null1>
k2	k1,1	k2,1	m2
k3	k1,1	k2,1	m3
k4	k1,1	k2,1	m4
k5	k1,1	k2,1	<null5>
k1	k1,2	k2,1	<null6>
k2	k1,2	k2,1	<null7>
k3	k1,2	k2,1	<null8>
k1	k1,2	k2,2	m9
k2	k1,2	k2,2	m10
k3	k1,2	k2,2	m11
k4	k1,2	k2,1	<null12>
k4	k1,2	k2,2	m13
k5	k1,2	k2,2	m14
....	....	....	.....

4.2.4 The Evaluation on the Proposed Missing Measure Data Handling Technique

For an illustration, the test data sample is obtained from a reference [13]. Moreover, the fact table in a user warehouse is partially tested and filled in with randomly missing data using the selected base dimension Time as shown in Figure 4.11. The choice of the base dimension Time is from the requirement of [39] for time-series data so that estimated results from [28], [39] and our technique are comparable.

TKey	PKey	CKey	Sale	Sale	Sale	Sale	Sale	Sale
1	1	1	2.851	2.851	2.851	2.851	2.851	2.851
2	1	1	2.672	2.672	2.672	2.672	2.672	2.672
3	1	1	2.755	<NULL>	3.73233333333333	2.6965	2.975	2.851
4	1	1	2.721	2.721	2.721	2.721	2.721	2.721
5	1	1	2.946	2.946	2.946	2.946	2.946	2.946
6	1	1	3.036	<NULL>	3.73233333333333	2.614	3.4708	2.851
7	1	1	2.282	2.282	2.282	2.282	2.282	2.282
8	1	1	2.212	<NULL>	3.73233333333333	4.202	1.4875	1.4255
9	1	1	2.922	2.922	2.922	2.922	2.922	2.922
10	1	1	4.301	4.301	4.301	4.301	4.301	4.301
11	1	1	5.764	5.764	5.764	5.764	5.764	5.764
12	1	1	7.132	7.132	7.132	7.132	7.132	7.132
13	1	1	2.541	<NULL>	3.748125	4.8035	2.851	2.851
14	1	1	2.475	2.475	2.475	2.475	2.475	2.475
15	1	1	3.031	3.031	3.031	3.031	3.031	3.031
16	1	1	3.266	3.266	3.266	3.266	3.266	3.266
17	1	1	3.776	<NULL>	3.748125	3.18666666666672	3.347	2.851
18	1	1	3.23	<NULL>	3.748125	3.10733333333339	3.4708	2.851
19	1	1	3.028	3.028	3.028	3.028	3.028	3.028
20	1	1	1.759	1.759	1.759	1.759	1.759	1.759
21	1	1	3.595	3.595	3.595	3.595	3.595	3.595
22	1	1	4.474	4.474	4.474	4.474	4.474	4.474
23	1	1	6.838	<NULL>	3.748125	5.2335	7.0655	6.41475
24	1	1	8.357	8.357	8.357	8.357	8.357	8.357

Actual Data | Randomly Missing Data | Filled with [28] | Filled with [39] | Filled with Individual Probabilities | Filled with Average Probabilities

Figure 4.11 The Results from the Filling-in-Missing-Data Techniques

The probabilities in a period of one year obtained from our two approaches are kept in our 5NF tables generated from the information warehouse conceptual meta schema in Figure 5.3 that will be introduced in the chapter 5. The two tables Table 4.7 and Table 4.8 show individual probabilities and average probabilities associated with the dimension Time calculated by using our technique.

For the comparison of our proposed technique with other current techniques, the Euclidean distance is studied to show how similar estimated data from all techniques are to the actual data. The smaller Euclidean distance implies the more similar estimated data to the actual data. Hence, the corresponding technique provides better expected results.

The Euclidean distance for estimated data using [K. Sattler, et al. - 28] is 3.8762.

The Euclidean distance for estimated data using [P. Jermyn, et al. - 39] is 3.4925.

The Euclidean distance for estimated data using average probabilities is 1.3916.

The Euclidean distance for estimated data using individual probabilities is 1.0734.

Table 4.7 Individual Probabilities

YearN	TKey	Individual Probability
1962	1	0.0606860158311346
1962	2	0.0527704485488127
1962	3	0.0633245382585752
1962	4	0.0686015831134565
1962	5	0.0712401055408971
1962	6	0.0738786279683377
1962	7	0.0633245382585752
1962	8	0.0316622691292876
1962	9	0.079155672823219
1962	10	0.0976253298153034
1962	11	0.150395778364116
1962	12	0.187335092348285
1963	13	0.0606860158311346
1963	14	0.0527704485488127
1963	15	0.0633245382585752
1963	16	0.0686015831134565
1963	17	0.0712401055408971
1963	18	0.0738786279683377
1963	19	0.0633245382585752
1963	20	0.0316622691292876
1963	21	0.079155672823219
1963	22	0.0976253298153034
1963	23	0.150395778364116
1963	24	0.187335092348285

Table 4.8 Average Probabilities

YearN	TKey	Average Probability
1962	1	0.0645161290322581
1962	2	0.0483870967741935
1962	3	0.0645161290322581
1962	4	0.0645161290322581
1962	5	0.0645161290322581
1962	6	0.0645161290322581
1962	7	0.0645161290322581
1962	8	0.032258064516129
1962	9	0.0806451612903226
1962	10	0.112903225806452
1962	11	0.145161290322581
1962	12	0.193548387096774
1963	13	0.0645161290322581
1963	14	0.0483870967741935
1963	15	0.0645161290322581
1963	16	0.0645161290322581
1963	17	0.0645161290322581
1963	18	0.0645161290322581
1963	19	0.0645161290322581
1963	20	0.032258064516129
1963	21	0.0806451612903226
1963	22	0.112903225806452
1963	23	0.145161290322581
1963	24	0.193548387096774

The evaluation on techniques with random missing data is graphically shown in Figure 4.12 and in Figure 4.13.

Obviously, the estimated measure data gained from our technique are closer to the actual measure data than one from other techniques. Furthermore, we believe that with the huge volume of warehouse data and the support from experienced statisticians, analysts, and warehouse administrators according to the nature of their data, users are able to select a suitable base dimension and provide the system with good sample data based on their business to tackle the problem with missing data in their warehouse. Therefore, our technique is applicable and useful to handle missing data in data warehouses as well as in temporal data warehouses.

4.3 Summary

This chapter has discussed two main focuses in the ETL process of a temporal information warehouse system. The first focus is on how to collect metadata of an operational database corresponding to a warehouse version in which users are interested for the loading task. The second one is on the treatment of missing data.

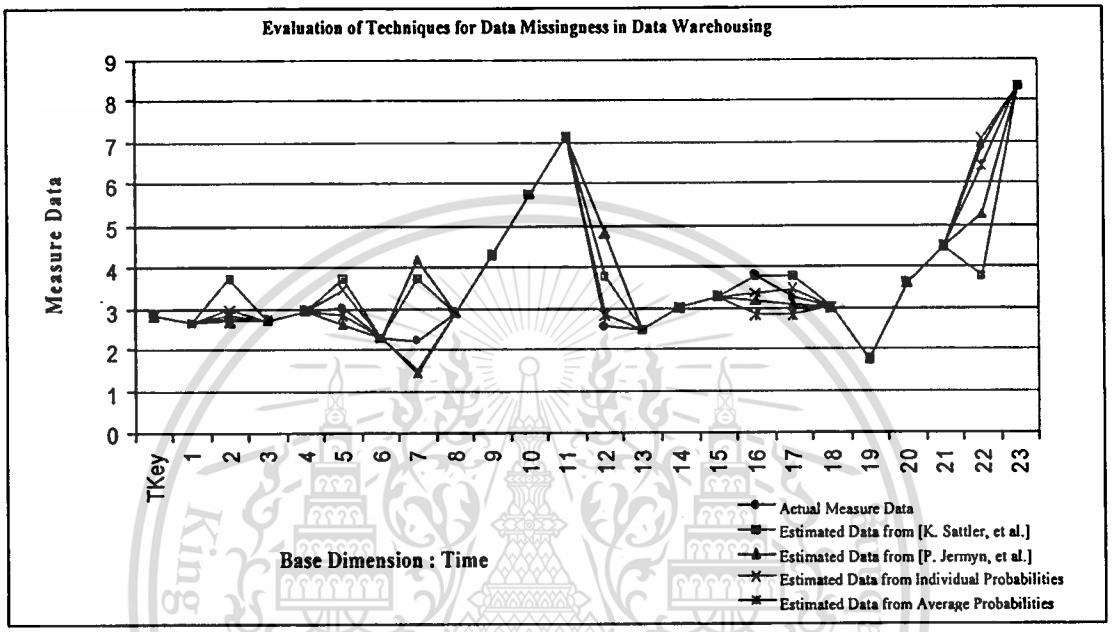


Figure 4.12 A Graph Showing Comparison of Filling-in-Missing-Data Techniques

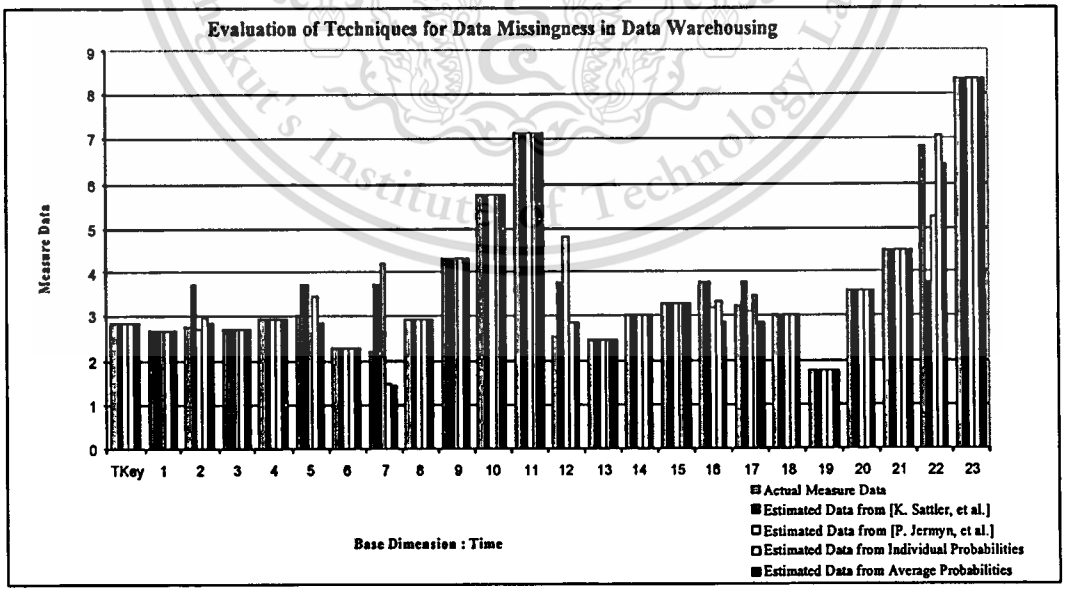


Figure 4.13 A Respective Chart Showing Comparison of Filling-in-Missing-Data Techniques

Firstly, expected source metadata are easily gathered by taking advantage of NIAM/ORM conceptual schemas that were formerly used to model operational sources at the conceptual level. Based on the fact-oriented nature of a NIAM/ORM conceptual schema, a deep-structured natural language interface was built to allow users to describe their NIAM/ORM conceptual schemas using simple English sentences.

Secondly, a novel technique applicable to fill in missing measure data in data warehouses with estimated values calculated by using probabilities and the K-means clustering technique has been presented. In comparison, the result from the proposed technique was found to be more accurate than those from other techniques. Besides, the technique is not only applicable to data warehouse systems but also to temporal data warehouse systems based on warehouse versioning. This technique was implemented in the ETL process of our TIWA1 architecture for the treatment of random missing data. As future research, the descriptive attribute values which are missing at the time the extraction is performed require more time and efforts because at this moment all techniques are unable to be applied on the descriptive attributes of dimension tables in data warehouses.

Chapter 5

The Design of a Warehouse Schema

Since 1995, star schemas have been popularly utilized to represent warehouse data at the logical level. Data warehouse design mainly focuses on the design of a warehouse star schema. There are many proposals on conceptual data warehouse design of a star schema based on some well-known data models such as the Entity Relationship Model (ERM), and an extension to the Unified Modeling Language (UML). This chapter presents the design of an information warehouse using the Nijssen's Information Analysis Methodology (NIAM) modeling, known as the Object Role Modeling (ORM) nowadays, for both user modeling of the warehouse and the underlying implementation model of the system. A warehouse multi-star/star schema is automatically and directly generated from the input schema description using a deep-structured natural language interface. Users input simple English sentences together with the structured sections for type declarations and constraints on these sentences. Since the input NIAM schema can also be transformed into 5NF operational relational schemas, this section allows warehouse users to generate star schemas from their operational schemas. This leads to a fact that mappings between operational sources and warehouses are able to be easily caught for the loading process.

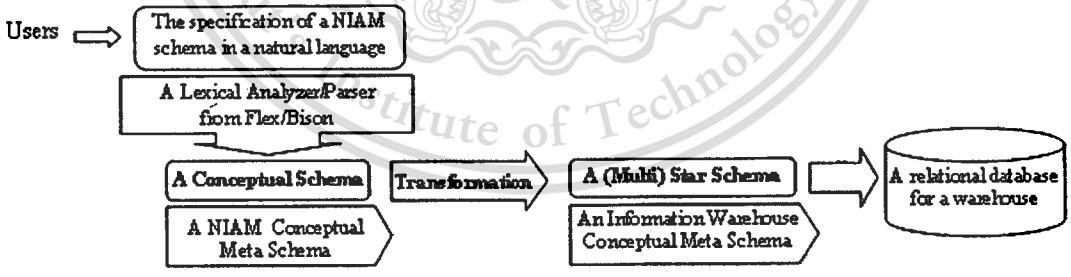


Figure 5.1 A Warehouse Schema Design Architecture

5.1 A Warehouse Schema Design Approach

Using the same methodology in [54], the warehouse schema design steps in Figure 5.1 is subdivided into three main tasks, namely, scanning/parsing the input from users, transforming the conceptual schema obtained after that, and generating fact structures

and dimensions for the user data warehouse from the (multi) star schema which is defined as an SQL script in Data Definition Language (DDL).

Firstly, for the task 'scanning/parsing the input from users', the Flex/Bison tool is employed to develop the lexical analyzer and the parser using a context free grammar. The lexical analyzer/parser performs on the specification of an NIAM model input into the system through the deep-structured natural language interface. The specification conformed to the pre-defined context free grammar includes natural language sentences describing fact types and reference types in terms of NIAM elements, some constraints on these sentences and data type declarations. In addition, information about the time validity of the model input is determined to establish a new version for a temporal data warehouse with schema versioning.

Secondly, once the operational conceptual schema is available in the NIAM meta tables generated from the NIAM conceptual meta schema in Figure 5.2, the schema transformation rules are automatically applied on the information about the operational conceptual schema in order to derive a corresponding (multi) star schema. During the schema transformation process, the output information about a (multi) star schema is gathered little by little and kept in other 5NF meta tables generated from the Information Warehouse Conceptual Meta Schema in Figure 5.3 until the output schema is completed. Apart from the warehouse schema information in the 5NF meta tables, an SQL script in DDL is also created.

Finally, from the SQL script in DDL in the previous task, fact tables and dimension tables are easily created using any existing database management system (DBMS) with a little syntax modification if necessary.

Note that these two conceptual schemas for the system, the NIAM conceptual meta schema in Figure 5.2 and the information warehouse conceptual meta schema in Figure 5.3, are designed by using the NIAM modeling too. The NIAM conceptual meta schema is used to generate 5NF meta tables in which meta data of NIAM operational conceptual schemas are managed. The latter will be used for the generation of other 5NF meta tables which contain not only meta data of user warehouse (multi) star schemas but also additional information about warehouse versions and storages in case that users are interested in temporal information warehouses. Summing up, the information warehouse

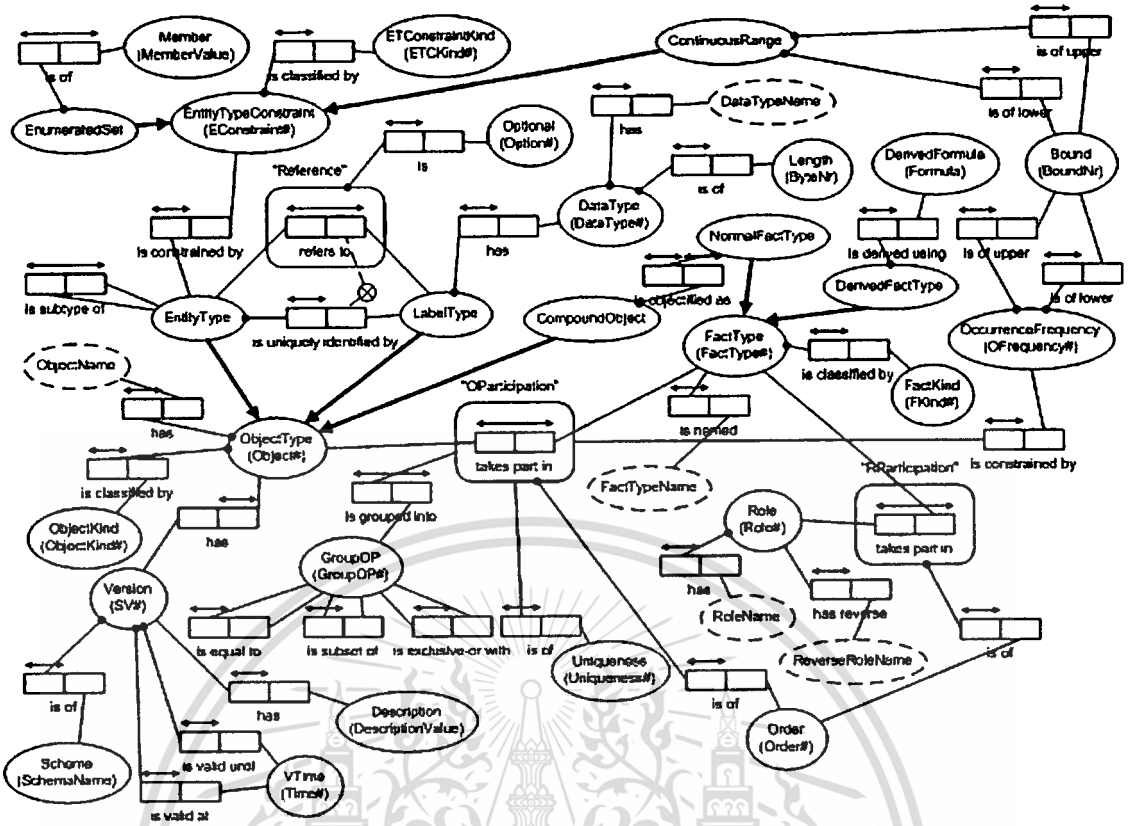


Figure 5.2 A NIAM Conceptual Meta Schema

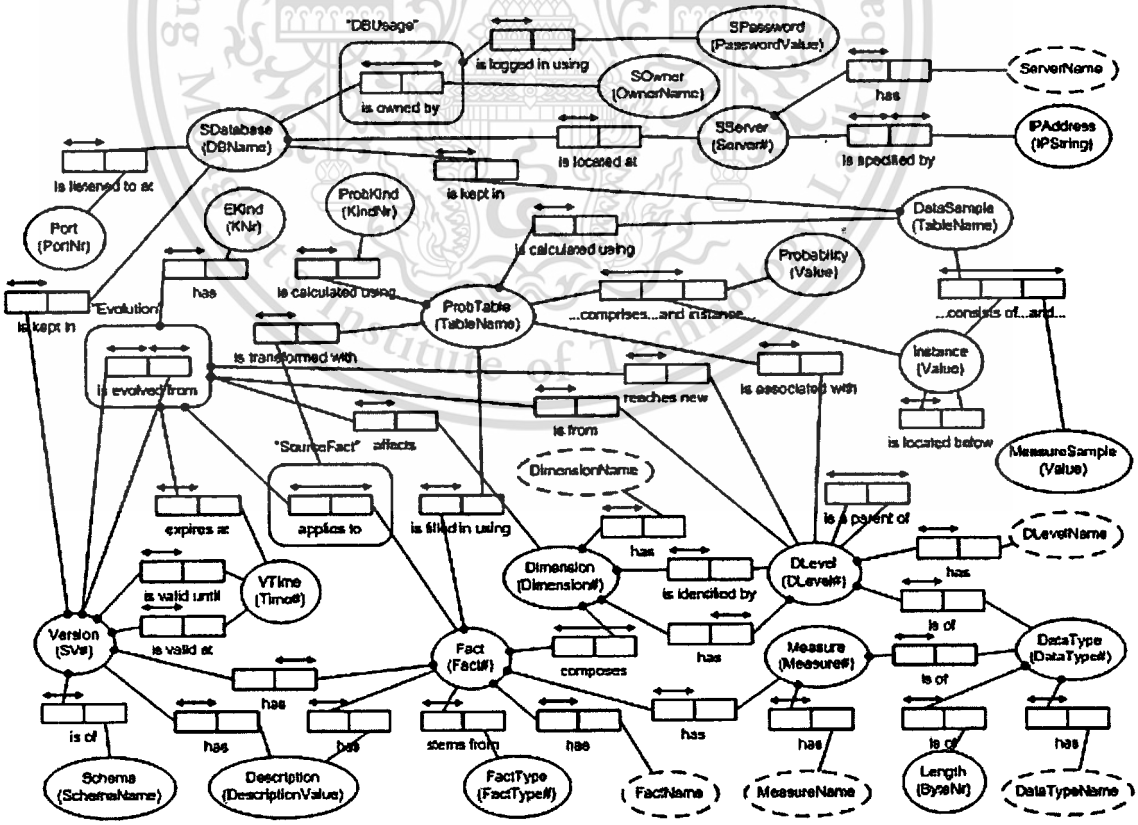


Figure 5.3 An Information Warehouse Conceptual Meta Schema

conceptual meta schema describes star schemas and other warehouse information parameters.

5.2 The Description of a Conceptual Schema in a Deep-structured Natural Language

The grammar in Figure 54 is a context-free grammar in the Backus-Naur form (BNF). uXXX are terminal symbols from user inputs, <xxx> are non-terminal symbols. XXX and xxx are reserved terminal symbols which are not allowed to be used as names of object types or roles. All reserved terminal symbols are case-sensitive.

```
<nschema> ::= <version> <facts> <constr> <dtype>
<version> ::= VERSION uNAME <description> <stime> <etime> END.
<description> ::= DESCRIPTION : uDESCRIPTION .
<stime> ::= START VALIDITY : uNUMBER / uNUMBER / uNUMBER .
<etime> ::= END VALIDITY  : uNUMBER / uNUMBER / uNUMBER .
<facts> ::= <unaryfacts>| <taryfacts>| <naryfacts>| <unaryfacts> <facts>
        | <taryfacts> <facts>| <naryfacts> <facts>
<unaryfacts> ::= uNAME uROLENAME.
<taryfacts> ::= uNAME uROLENAME / uROLENAME uNAME .
        | uNAME  : uNAME uROLENAME / uROLENAME uNAME .
<naryfacts> ::= uNAME  : <nfacttype> . ( <nconstraint> ) IS UNIQUE .
<nfacttype> ::= uNAME uROLENAME uNAME uROLENAME uNAME
        | <nfacttype> uROLENAME uNAME
<nconstraint> ::= uNAME  uNAME | <nconstraint> uNAME
<constr> ::= CONSTRAINT <identifier> <unique_optional> <entity_section> <subtype_section>
<frequency_section> <equal_section> <exclusive_section> <subset_section> END .
<identifier> ::= uNAME IS UNIQUELY IDENTIFIED BY uNAME : <datatype> ( uNUMBER ) .
        | uNAME IS UNIQUELY IDENTIFIED BY uNAME  : <datatype> ( uNUMBER ) .
<identifier>
<unique_optional> ::= EACH uNAME uROLENAME <quantity> uNAME .
        | EACH uNAME uROLENAME <quantity> uNAME . <unique_optional>
<quantity> ::= EXACTLY ONE | ZERO OR ONE | ZERO OR MORE | ONE OR MORE
<entity_section>::= /*empty*/ | ENTITY <entity>
<entity> ::= uNAME { <enumeratedset> } . | uNAME { <enumeratedset> } . <entity>
        | uNAME [ <continuousrange> ] . | uNAME [ <continuousrange> ] . <entity>
<enumeratedset> ::= uSTRING , uSTRING | uNUMBER , uNUMBER
        | uSTRING , <enumeratedset> | uNUMBER , <enumeratedset>
<continuousrange> ::= uNUMBER . . uNUMBER
<subtype_section> ::= /*empty*/ | SUBTYPE <subtype>
<subtype> ::= uNAME = <category> . | uNAME = <category> . <subtype>
<category> ::= uNAME uROLENAME uNAME WITH uNAME <comparison> uSTRING
        | uNAME uROLENAME uNAME WITH uNAME <comparison> uNUMBER
        | UNION ( <category> , <category> )
        | INTERSECT ( <category> , <category> )
        | MINUS ( <category> , <category> )
<comparison> ::= > | < | >= | <= | = | <>
<frequency_section> ::= /*empty*/ | FREQUENCY <frequency>
<frequency> ::= uNAME  : ( <objectname> ) OCCURS uNUMBER TIMES .
        | uNAME  : ( <objectname> ) OCCURS uNUMBER TIMES . <frequency>
        | uNAME  : ( <objectname> ) OCCURS AT LEAST uNUMBER AND AT MOST
uNUMBER TIMES .
        | uNAME  : ( <objectname> ) OCCURS AT LEAST uNUMBER AND AT MOST
uNUMBER TIMES . <frequency>
<objectname> ::= uNAME | uNAME , <objectname>
```

```
<equal_section> ::= /*empty*/ | EQUALITY <equal>
<equal> ::= FOR EACH uNAME : uROLENAME ( uNAME ) IFF uROLENAME ( uNAME ) .
        | FOR EACH uNAME : uROLENAME ( uNAME ) IFF uROLENAME ( uNAME ) . <equal>
<exclusive_section> ::= /*empty*/ | EXCLUSIVITY <exclusive>
<exclusive> ::= NO uNAME PLAYS BOTH uROLENAME ( uNAME ) AND uROLENAME (
uNAME ) .
        | NO uNAME PLAYS BOTH uROLENAME ( uNAME ) AND uROLENAME ( uNAME ) .
<exclusive>
<subset_section> ::= /*empty*/ | SUBSET <subset>
<subset> ::= FOR EACH uNAME : IF uROLENAME ( uNAME ) THEN uROLENAME ( uNAME ) .
        | FOR EACH uNAME : IF uROLENAME ( uNAME ) THEN uROLENAME ( uNAME ) .
<subset>
<dtype> ::= DATATYPE <declaration> END .
<declaration> ::= uNAME : <datatype> ( uNUMBER ) . | uNAME : <datatype> ( uNUMBER ) .
<declaration>
<datatype> ::= bigint | binary | char | datetime | decimal | float | int | money | nchar | numeric | nvarchar |
real | smallint | varchar
```

Figure 5.4 The Context-Free Grammar for a NIAM Conceptual Schema Specification

uXXX includes uDESCRIPTION, uNAME, uROLENAME, uNUMBER, and uSTRING. Each of them is in turn described at the lexical part as presented in Table 5.1. All user terminals must be uniquely identified in the same category to which they belong.

Table 5.1 List of User Terminal Symbols

User Terminal uXXX	Description	Example
uDESCRIPTION	"\""[a-zA-Z0-9#]+"\""	"FoodMart2000"
uNAME	[A-Z][a-zA-Z#0-9_-]+	ConceptualSchemaTest2, Sale1
uROLENAME	[a-z][a-zA-Z#0-9_-]+	belongs_to, is_of, has, at
uNUMBER	[0-9]+	01, 1995, 9999
uSTRING	"'"[a-zA-Z0-9]+"'"	'temporal', 'Schema', 'Quarter1'

Non-terminal symbols (variables) <xxx> consists of <nschema>, <version>, <facts>, <constr>, <dtype>, <description>, <stime>, <etime>, <unaryfacts>, <taryfacts>, <naryfacts>, <nfacttype>, <nconstraint>, <identifier>, <unique_optional>, <entity_section>, <subtype_section>, <frequency_section>, <equal_section>, <exclusive_section>, <subset_section>, <datatype>, <quantity>, <entity>, <enumeratedset>, <continuousrange>, <subtype>, <category>, <comparison>, <frequency>, <objectname>, <equal>, <exclusive>, <subset>, <declaration>.

Structural reserved terminal symbols are listed as VERSION, END, DESCRIPTION, START, VALIDITY, IS, UNIQUE, CONSTRAINT, UNIQUELY, IDENTIFIED, BY, EACH, EXACTLY, ONE, ZERO, OR, MORE, ENTITY, SUBTYPE, WITH, UNION, INTERSECT, MINUS, FREQUENCY, OCCURS, TIMES, AT, LEAST, AND, MOST, EQUALITY, FOR, IFF, EXCLUSIVITY, NO, PLAYS, BOTH, SUBSET, IF, THEN, EACH, DATATYPE.

And, data reserved terminal symbols for data type declarations are bigint, binary, char, datetime, decimal, float, int, money, nchar, numeric, nvarchar, real, smallint, varchar.

Using above non-terminal symbols and terminal symbols according to the grammar, the textual specification of a conceptual schema in NIAM is of four parts, they are <version>, <facts>, <constr>, and <dtype>.

In the <version> part, users give our system brief information about their conceptual schema including a schema name, a short description, and an interval of valid time in the case that their warehouse system is going to be developed with schema versioning.

In the <facts> part, the representation of the Universe of Discourse (UoD) is given. This part is the heart of the user input. The representation is deep-structured natural language sentences representing relationships among many objects in the UoD. Unary fact types that show properties of involved objects are specified from the statements in the <unaryfacts> part. Binary fact types and reference types which are specified from the <taryfacts> part are obtained directly from the statements in which two objects, entity types or label types are included. N-ary fact types with N greater than 2 specified from the <naryfacts> parts are gained from the statements in the <nfacttype> part which are followed by the uniqueness constraints on objects involved in these statements in the <nconstraint> part. Moreover, objectifying relationships is done with object names uNAME followed by colons ":" at the beginning of the statements in the <taryfacts> and <naryfacts> parts if users wish to use nested objects.

In the <constr> part, users add constraints on objects in the UoD and on relationships among objects in the UoD. Several kinds of constraints supported by our system are uniqueness constraint, mandatory-optional role <unique_optional>, entity type constraint <entity_section>, subtype constraint <subtype_section>, occurrence

frequency constraint <frequency_section>, equal set constraint <equal_section>, subset constraint <subset_section>, and exclusive set constraint <exclusive_section>. These constraint declarations are quite useful for the description of an operational conceptual schema. Therefore, we use the grammar for modeling operational sources as well. In addition, the declarations of identifiers of entity types are included in this part. The data type declarations of these identifiers follow them in the <datatype> part right after the colon.

In the <dtype> part, users specify which objects in the UoD are labels and which data types those labels are associated with.

In summary, abiding by our grammar, users input simple English sentences describing a NIAM conceptual schema. Due to the simplicity of the deep-structured natural language which has been defined, this approach is more expressive than existing approaches based on the ERM [11, 29, 33, and 36], the UML [24, 25, 49, 50, and 51] or the entity-based model language [29].

5.3 Mappings from a Conceptual Schema to a Warehouse Schema

After parsed and checked for the semantics, the user input is redefined in terms of NIAM elements such as entity types, label types, fact types, constraints, etc., which form a conceptual schema. All information about the operational conceptual schema is saved into 5NF meta tables generated from the NIAM conceptual meta schema. Though most kinds of constraints supported in the fact-oriented approach are described in the NIAM conceptual meta schema, these constraints are captured in the user input, but not implemented because the warehouse schema design focuses on the extraction of fact structures and dimensions with their hierarchies. Besides, the information about time, and the description of the NIAM schema obtained from the user input forms a schema version so that it is able to keep track of the schema evolution over time.

To gather information about the result warehouse schema, transformation rules are automatically performed on the information about the input conceptual schema. After the schema transformation, a corresponding warehouse (multi) star schema including fact structures, dimensions and all descriptive attributes and hierarchies of these dimensions is reached. The following are transformation rules from a conceptual schema which

models a (multi) star schema to a corresponding (multi) star schema based on the definition of a warehouse schema given in the section 2 and the purpose of our approach is to produce a structure available to keep data from operational sources and available to be further processed. Hence, each star structure in the (multi) star schema has one fact structure with one measure attribute so that it well matches an online analytical processing (OLAP) cube to be further processed by some existing front-end tools.

From the information about the input conceptual schema, crucial elements including entity types, label types, compound objects, reference types, identifiers for entity types, binary fact types, n-ary fact types for n above two, all uniqueness constraints on fact types, and each data type of every identifier are determined. The transformation from a NIAM conceptual schema into a warehouse schema is as follows:

For each n-ary fact type with n greater than two, follow the following steps to form fact structures and dimensions with their descriptive attributes and hierarchies.

Step 1: Find an object type which plays a role in the n-ary fact type of no effect from the uniqueness constraints and involved in the n-ary fact type, which is being processed. Proceed to step 2.

Step 2: Create a potential fact structure.

If the object type is an entity type and its identifier has a data type suitable for aggregations, then create a new fact structure with one measure attribute that is the entity type, and go to step 3 to create dimensions associated with the fact structure. Otherwise, return to step 1.

Step 3: Create dimensions.

For each object type playing the role that is uniquely constrained and involved in the n-ary fact type, which is being processed, create a dimension corresponding to the object type.

Step 3.1: Specify the key of a dimension.

If the object type is an entity type, then the identifier that uniquely identifies the entity type becomes the key of the dimension. Otherwise, the object type is a compound object formed from another n - ary fact type for n >= 2 with the uniqueness constraint

spanning all roles in the fact type. The key of the fact type from which the compound object stemmed becomes the key of the dimension. Mark the key of the dimension which has been specified as a current dimension level in the case of an entity type and mark all components of the key of the dimension as current dimension levels in the case of a compound object.

Step 3.2: Derive all descriptive attributes of a dimension level.

For each current dimension level at the previous step, find all descriptive attributes which are label types from reference types which are related to the entity type uniquely identified by the current dimension level.

Step 3.3: Derive all hierarchies of a dimension.

Find all next dimension levels following the current dimension level from binary fact types related to the entity type uniquely identified by the current dimension level.

For each binary fact type which is related to the entity type uniquely identified by the current dimension level, check whether or not the other object type is suitable to become a next dimension level of the current dimension level. It is suitable if the binary fact type is an m:1 relationship (as shown in the form presented in Figure 5.5) and the object type B on the many side is different from all object types involved in the n-ary fact type, which is being processed. If B is an entity type, then its identifier becomes a next dimension level of the current dimension level. If B is a compound object, all components of the key of the fact type from which B is formed become next dimension levels of the current dimension level.

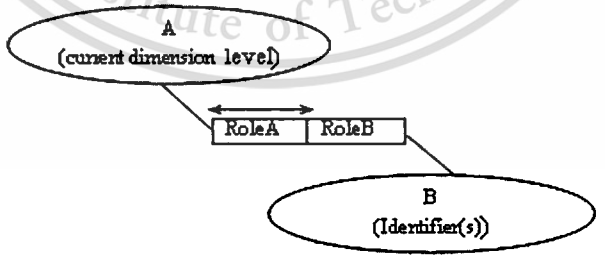


Figure 5.5 The Expected Form of a Binary Fact Type

Establish hierarchical relationships between the current dimension level and all of its next dimension levels. Mark these next dimension levels as current dimension levels, and then return to step 3.2.

The two steps 3.2 and 3.3 are repeated until there is no more dimension level marked as a current dimension level.

Step 3.4: Put all key, descriptive attributes, and dimension levels together to form a complete dimension.

Step 4: Define the key of the fact structure. It is a composition of all attributes each of which refers to the key of a dimension associated with the fact structure. One star structure has been completely derived for the target warehouse (multi) star schema.

Step 5: Return to step 1 for another fact structure until there is no more n-ary fact type for n > 2

After a (multi) star schema is available in the 5NF warehouse meta tables in the form of fact structures, dimensions, and all descriptive attributes and hierarchies of each dimension, an SQL script in DDL is created to generate all table structures for the user data warehouse.

5.4 An Example on the Design of a Warehouse Schema

The example is from FoodMart 2000, an example of Microsoft Analysis Services 2000. FoodMart 2000 is a database formatted in an Access file for analysis services. The example on the design of a warehouse schema is built with the following steps:

- Modeling FoodMart 2000 using the NIAM modeling to obtain a textual description in forms of our predefined grammar
- The output of the transformation is a relational schema for the FoodMart 2000 warehouse by using Microsoft SQL Server 2000

5.4.1 Input

```
VERSION CSFoodMart2000
        DESCRIPTION : "FoodMart2000".
        START VALIDITY : 01/01/1997.
        END VALIDITY : 12/31/9999.
END.

Account is_of/has AccountType.
Account has/describes Description.
City is_in/has District.
Customer opened_an_account_on/is_for Date.
Customer is_of/has Gender.
Customer has_an/is_paid_for IncomeAmount.
Customer is_at/is_with Location.
Customer is_with/is_for Marriage.
Customer has_a_first_name_of/belongs_to FirstName.
Customer has_a_last_name_of/belongs_to LastName.
```

Customer stays_at/is_for Place.
District belongs_to/has StateProvince.
Location is_in/has City.
Product is_of/has Brand.
Product belongs_to/has ProductClass.
Product has/belongs_to ProductName.
Product has/belongs_to RetailPrice.
Product has/belongs_to StockNumber.
ProductClass is_of/is ProductSubCategory.
ProductCategory belongs_to/has ProductDepartment.
ProductDepartment belongs_to/has ProductFamily.
ProductSubCategory is_classified_into/includes ProductCategory.
Promotion is_of/is_for Amount.
Promotion is_effective_on/is_for EffectiveDate.
Promotion is_ended_on/is_for EndedDate.
Promotion uses/is_used_by MediaType.
Promotion has/is_for PromotionName.
Region belongs_to/has Country.
StateProvince belongs_to/has Region.
Store is_located_at/has Location.
Store stays_at/belongs_to SPlace.
Store has/belongs_to StoreName.
Store is_managed_by/manages StoreManager.
Store is_of/has StoreType.
TimeByDay is_on/is Date.
TimeByDay is_on/is TheDay.
TimeByDay is_in/includes MonthOfYear.
TimeByDay is_in/includes WeekOfYear.
MonthOfYear is_the_th/is_for Month.
MonthOfYear is_in/includes QuarterOfYear.
MonthOfYear has/is_for TheMonth.
QuarterOfYear is_in/includes Year.
QuarterOfYear is_the_th/is_for Quarter.
WeekOfYear is_the_th/is_for Week.
WeekOfYear is_in/includes Year.
Warehouse is_at/is_with Location.
Warehouse is_owned_by/owns Owner.
Warehouse stays_at/has WPlace.
Warehouse belongs_to/includes WarehouseClass.
Warehouse has/belongs_to WarehouseName.
WarehouseClass has/describes WDescription.
Expense : Account of ExpenseAmount is_used_for Store on TimeByDay.
(Account Store TimeByDay) IS UNIQUE.
InventorySales : Warehouse serves Store for Product on TimeByDay with_the_sales_of InventoryAmount.
(Warehouse Store Product TimeByDay) IS UNIQUE.
Sales : Customer buys Product of_the SalesAmount with Promotion at Store on TimeByDay.
(Customer Product Promotion Store TimeByDay) IS UNIQUE.

CONSTRAINT
Account IS UNIQUELY IDENTIFIED BY AccountId : int(4).
AccountType IS UNIQUELY IDENTIFIED BY AccType : varchar(20).
Description IS UNIQUELY IDENTIFIED BY ADescription : nvarchar(50).
ExpenseAmount IS UNIQUELY IDENTIFIED BY AExpense : float(8).
Store IS UNIQUELY IDENTIFIED BY StoreId : int(4).
TimeByDay IS UNIQUELY IDENTIFIED BY TimeId : int(4).
City IS UNIQUELY IDENTIFIED BY CityName : nvarchar(50).
District IS UNIQUELY IDENTIFIED BY DistrictName : nvarchar(50).
Customer IS UNIQUELY IDENTIFIED BY CustomerId : int(4).
Date IS UNIQUELY IDENTIFIED BY TheDate : datetime(8).

Gender IS UNIQUELY IDENTIFIED BY GenderCode : nvarchar(1).
IncomeAmount IS UNIQUELY IDENTIFIED BY AIncome : money(8).
Location IS UNIQUELY IDENTIFIED BY LocationId : int(4).
Marriage IS UNIQUELY IDENTIFIED BY MaritalStatus : nvarchar(1).
Place IS UNIQUELY IDENTIFIED BY Address : nvarchar(100).
Product IS UNIQUELY IDENTIFIED BY ProductId : int(4).
SalesAmount IS UNIQUELY IDENTIFIED BY ASales : float(8).
Promotion IS UNIQUELY IDENTIFIED BY PromotionId : int(4).
Brand IS UNIQUELY IDENTIFIED BY BrandName : nvarchar(50).
ProductClass IS UNIQUELY IDENTIFIED BY ProductClassId : int(4).
RetailPrice IS UNIQUELY IDENTIFIED BY ARetailPrice : money(8).
StockNumber IS UNIQUELY IDENTIFIED BY AStockNumber : float(8).
ProductSubCategory IS UNIQUELY IDENTIFIED BY ProductSubCategoryId : nvarchar(15).
ProductCategory IS UNIQUELY IDENTIFIED BY ProductCategoryId : nvarchar(15).
ProductDepartment IS UNIQUELY IDENTIFIED BY ProductDepartmentId : nvarchar(15).
ProductFamily IS UNIQUELY IDENTIFIED BY ProductFamilyId : nvarchar(15).
Amount IS UNIQUELY IDENTIFIED BY AmountMoney : money(8).
EffectiveDate IS UNIQUELY IDENTIFIED BY EffDate : datetime(8).
EndedDate IS UNIQUELY IDENTIFIED BY EndDate : datetime(8).
MediaType IS UNIQUELY IDENTIFIED BY MediaTypeP : nvarchar(255).
Region IS UNIQUELY IDENTIFIED BY RegionName : nvarchar(50).
Country IS UNIQUELY IDENTIFIED BY CountryName : nvarchar(50).
StateProvince IS UNIQUELY IDENTIFIED BY StateName : nvarchar(50).
SPlace IS UNIQUELY IDENTIFIED BY SAddress : nvarchar(100).
StoreManager IS UNIQUELY IDENTIFIED BY SManagerName : nvarchar(255).
StoreType IS UNIQUELY IDENTIFIED BY SType : nvarchar(255).
MonthOfYear IS UNIQUELY IDENTIFIED BY YMonth : nvarchar(8).
Month IS UNIQUELY IDENTIFIED BY MonthNo: smallint(2).
QuarterOfYear IS UNIQUELY IDENTIFIED BY YQuarter : nvarchar(8).
Quarter IS UNIQUELY IDENTIFIED BY QuarterNo: nvarchar(2).
Year IS UNIQUELY IDENTIFIED BY TheYear : nvarchar(4).
WeekOfYear IS UNIQUELY IDENTIFIED BY YWeek: nvarchar(8).
Week IS UNIQUELY IDENTIFIED BY WeekNo: smallint(2).
Warehouse IS UNIQUELY IDENTIFIED BY WarehouseId : int(4).
Owner IS UNIQUELY IDENTIFIED BY OwnerName : nvarchar(50).
InventoryAmount IS UNIQUELY IDENTIFIED BY AInventory : float(8).
WPlace IS UNIQUELY IDENTIFIED BY WAddress : nvarchar(255).
WarehouseClass IS UNIQUELY IDENTIFIED BY WarehouseClassId : int(4).
WDescription IS UNIQUELY IDENTIFIED BY WHDescription : nvarchar(255).

EACH Account is_of EXACTLY ONE AccountType.
EACH AccountType has ZERO OR MORE Account.
EACH Account has ZERO OR ONE Description.
EACH Description describes ZERO OR MORE Account.
EACH City is_in EXACTLY ONE District.
EACH District has ONE OR MORE City.
EACH Customer opened_an_account_on ZERO OR ONE Date.
EACH Date is_for ZERO OR MORE Customer.
EACH Customer is_of EXACTLY ONE Gender.
EACH Gender has ONE OR MORE Customer.
EACH Customer has_an EXACTLY ONE IncomeAmount.
EACH IncomeAmount is_paid_for ZERO OR MORE Customer.
EACH Customer is_at EXACTLY ONE Location.
EACH Location is_with ZERO OR MORE Customer.
EACH Customer is_with EXACTLY ONE Marriage.
EACH Marriage is_for ZERO OR MORE Customer.
EACH Customer has_a_first_name_of EXACTLY ONE FirstName.
EACH FirstName belongs_to ZERO OR MORE Customer.
EACH Customer has_a_last_name_of EXACTLY ONE LastName.
EACH LastName belongs_to ZERO OR MORE Customer.

EACH Customer stays_at EXACTLY ONE Place.
EACH Place is_for ZERO OR MORE Customer.
EACH District belongs_to EXACTLY ONE StateProvince.
EACH StateProvince has ONE OR MORE District.
EACH Location is_in EXACTLY ONE City.
EACH City has ONE OR MORE Location.
EACH Product is_of EXACTLY ONE Brand.
EACH Brand has ZERO OR MORE Product.
EACH Product belongs_to EXACTLY ONE ProductClass.
EACH ProductClass has ZERO OR MORE Product.
EACH Product has EXACTLY ONE ProductName.
EACH ProductName belongs_to ZERO OR MORE Product.
EACH Product has EXACTLY ONE RetailPrice.
EACH RetailPrice belongs_to ZERO OR MORE Product.
EACH Product has EXACTLY ONE StockNumber.
EACH StockNumber belongs_to ZERO OR MORE Product.
EACH ProductClass is_of EXACTLY ONE ProductSubCategory.
EACH ProductSubCategory is ZERO OR MORE ProductClass.
EACH ProductCategory belongs_to EXACTLY ONE ProductDepartment.
EACH ProductDepartment has ZERO OR MORE ProductCategory.
EACH ProductDepartment belongs_to EXACTLY ONE ProductFamily.
EACH ProductFamily has ZERO OR MORE ProductDepartment.
EACH ProductSubCategory is_classified_into EXACTLY ONE ProductCategory.
EACH ProductCategory includes ZERO OR MORE ProductSubCategory.
EACH Promotion is_of EXACTLY ONE Amount.
EACH Amount is_for ZERO OR MORE Promotion.
EACH Promotion is_effective_on EXACTLY ONE EffectiveDate.
EACH EffectiveDate is_for ZERO OR MORE Promotion.
EACH Promotion is_ended_on EXACTLY ONE EndedDate.
EACH EndedDate is_for ZERO OR MORE Promotion.
EACH Promotion uses ZERO OR ONE MediaType.
EACH MediaType is_used_by ZERO OR MORE Promotion.
EACH Promotion has EXACTLY ONE PromotionName.
EACH PromotionName is_for ZERO OR MORE Promotion.
EACH Region belongs_to EXACTLY ONE Country.
EACH Country has ONE OR MORE Region.
EACH StateProvince belongs_to EXACTLY ONE Region.
EACH Region has ONE OR MORE StateProvince.
EACH Store is_located_at EXACTLY ONE Location.
EACH Location has ZERO OR MORE Store.
EACH Store stays_at EXACTLY ONE SPlace.
EACH SPlace belongs_to ZERO OR MORE Store.
EACH Store has EXACTLY ONE StoreName.
EACH StoreName belongs_to ZERO OR MORE Store.
EACH Store is_managed_by EXACTLY ONE StoreManager.
EACH StoreManager manages ZERO OR MORE Store.
EACH Store is_of EXACTLY ONE StoreType.
EACH StoreType has ZERO OR MORE Store.
EACH TimeByDay is_on EXACTLY ONE Date.
EACH Date is ONE OR MORE TimeByDay.
EACH TimeByDay is_on EXACTLY ONE TheDay.
EACH TheDay is ONE OR MORE TimeByDay.
EACH TimeByDay is_in EXACTLY ONE MonthOfYear.
EACH MonthOfYear includes ONE OR MORE TimeByDay.
EACH MonthOfYear is_the_th EXACTLY ONE Month.
EACH Month is_for ONE OR MORE MonthOfYear.
EACH MonthOfYear is_in EXACTLY ONE QuarterOfYear.
EACH QuarterOfYear includes ONE OR MORE MonthOfYear.
EACH MonthOfYear has EXACTLY ONE TheMonth.
EACH TheMonth is_for ONE OR MORE MonthOfYear.

```
        EACH QuarterOfYear is_in EXACTLY ONE Year.
        EACH Year includes ONE OR MORE QuarterOfYear.
        EACH QuarterOfYear is_the_th EXACTLY ONE Quarter.
        EACH Quarter is_for ONE OR MORE QuarterOfYear.
        EACH TimeByDay is_in EXACTLY ONE WeekOfYear.
        EACH WeekOfYear includes ONE OR MORE TimeByDay.
        EACH WeekOfYear is_the_th EXACTLY ONE Week.
        EACH Week is_for ONE OR MORE WeekOfYear.
        EACH WeekOfYear is_in EXACTLY ONE Year.
        EACH Year includes ONE OR MORE WeekOfYear.
        EACH Warehouse is_at EXACTLY ONE Location.
        EACH Location is_with ZERO OR MORE Warehouse.
        EACH Warehouse is_owned_by EXACTLY ONE Owner.
        EACH Owner owns ZERO OR MORE Warehouse.
        EACH Warehouse stays_at EXACTLY ONE WPlace.
        EACH WPlace has ZERO OR MORE Warehouse.
        EACH Warehouse belongs_to EXACTLY ONE WarehouseClass.
        EACH WarehouseClass includes ZERO OR MORE Warehouse.
        EACH Warehouse has EXACTLY ONE WarehouseName.
        EACH WarehouseName belongs_to ZERO OR MORE Warehouse.
        EACH WarehouseClass has EXACTLY ONE WDescription.
        EACH WDescription describes ZERO OR MORE WarehouseClass.
END.

DATATYPE
        FirstName : nvarchar(100).
        LastName : nvarchar(100).
        ProductName : nvarchar(255).
        PromotionName : nvarchar(100).
        StoreName : nvarchar(255).
        TheDay: nvarchar(15).
        TheMonth: nvarchar(15).
        WarehouseName : nvarchar(100).
END.
```

Figure 5.6 The Textual Description of the NIAM Conceptual Schema Modeling the FoodMart 2000

The NIAM conceptual schema modeling the FoodMart 2000 example corresponding to the above textual description is below presented in Figure 5.7 by making use of Microsoft Visio for Enterprise Architect. There are three n-ary fact types, namely Expense for the 4-ary fact type, InventorySales for the 5-ary fact type, and Sales for the 6-ary fact type.

5.4.2 Process

Abiding by the transformation rules, a corresponding multistar schema for the FoodMart 2000 warehouse is generated step-by-step as follows.

The process is here demonstrated for the 4-ary fact type InventorySales. Perform the same process for other n-ary fact types for n above two.

Step 1: Find ExpenseAmount as an object type playing a role in the 4ary fact type without effect of the uniquess constraint.

Step 2: Create a fact structure named as Fact_0_V39 with a measure attribute ExpenseAmount that is previously determined.

Step 3: For each object type, Store, Account, and TimeByDay, which is uniquely constrained in the 4-ary fact type, create a dimension, Store, Account, and TimeByDay, respectively if it doesn't exist. Consider the dimension Store.

Step 3.1: Specify the key of the dimension Store.

Store is an object type uniquely identified by StoreId. So StoreId is determined as the key of the dimension Store.

Step 3.2: Derive all descriptive attributes of the dimension level StoreId, which is the first level marked as a current dimension level. Determine StoreName as a descriptive attribute for the level StoreId from the reference type { Store has/belongs_to StoreName.} where StoreName is a label type.

Step 3.3: Derive all hierarchies of the dimension Store.

Determine Location(LocationId) as a next dimension level from the binary fact type { Store is_located_at/has Location. } that satisfies the form in Figure 55 from the uniqueness constraint { EACH Store is_located_at EXACTLY ONE Location. EACH Location has ZERO OR MORE Store.} where LocationId is the identifier of the object type Location. Establish a hierarchical relationship between StoreId and Location(LocationId).

Also, determine SPlace (Address), StoreManager (SManagerName), and StoreType(SType) as next dimension levels of the level StoreId.

Mark all next dimension levels of the level StoreId as current dimension level.

Repeat step 3.2 and 3.3 until there is no more dimension level marked as a current level.

Step 3.4: Complete the dimension Store with its key, descriptive attributes, dimension levels and hierarchies.

Step 4: Define the key of the fact structure Fact_0_V39. It is a composition of {StoreId, AccountId, TimeId} from dimensions {Store, Account, TimeByDay}, respectively.

Step 5: Return step 1 for another n-ary fact type (n>2).

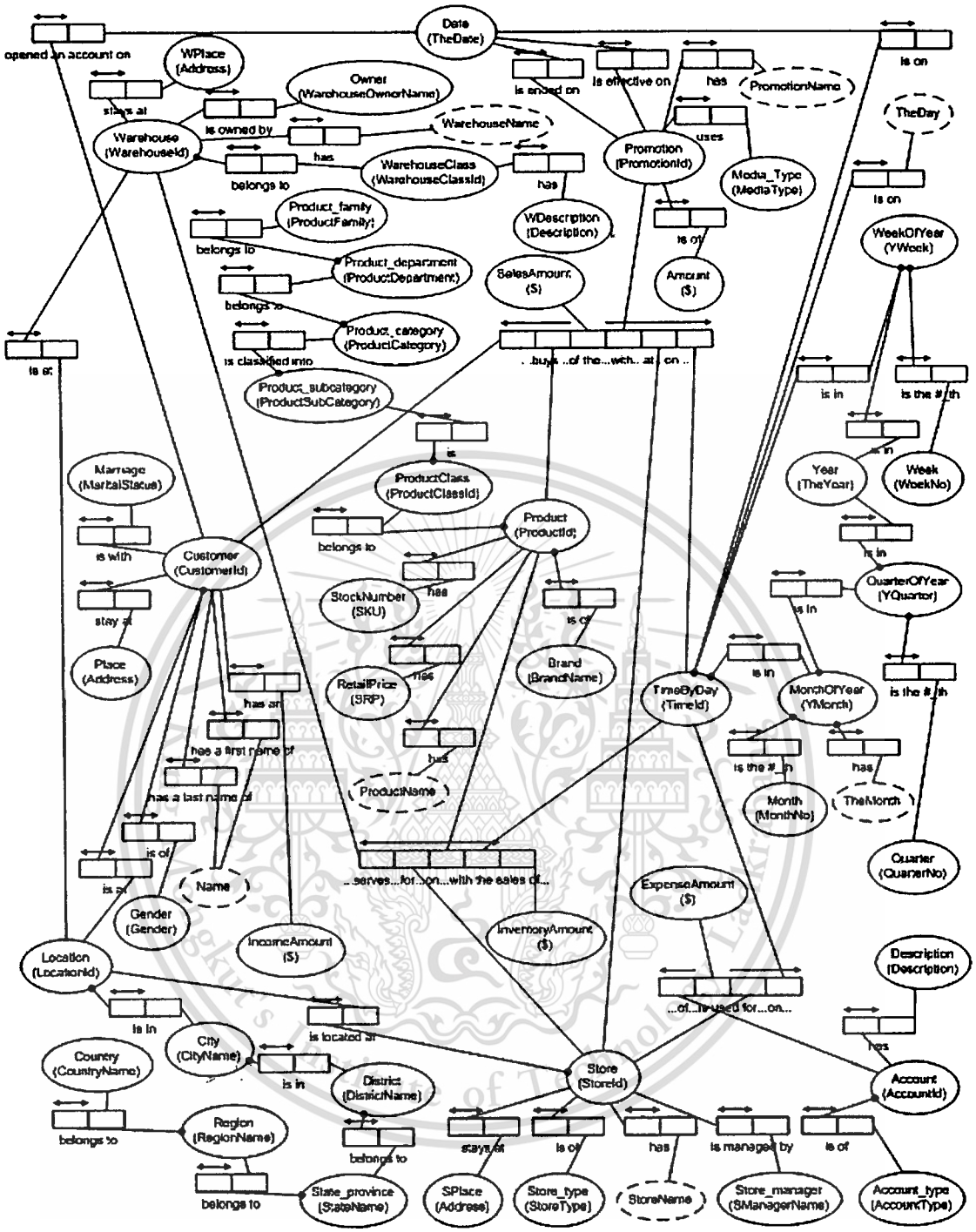


Figure 5.7 The Visualized NIAM Conceptual Schema Modeling the FoodMart 2000 Example

5.4.3 Output

The warehouse schema output in Figure 59 is a multistar schema of three fact structures, namely, Fact_0_V39, Fact_1_V39, and Fact_2_V39 corresponding to the 4

ary fact type Expense, 5-ary fact type InventorySales, and 6-ary fact type Sales. They share some dimensions Store_1_V39, TimeByDay_2_V39, and Product_4_V39. In addition, a corresponding SQL script in DDL is produced in Figure 5.8.

```
CREATE TABLE Account_0_V39
(
AccountId int PRIMARY KEY,
AccountType varchar (20),
Description nvarchar (50)
);
CREATE TABLE Store_1_V39
(
StoreId int PRIMARY KEY,
StoreName nvarchar (255),
Location int,
SPlace nvarchar (100),
StoreManager nvarchar (255),
StoreType nvarchar (255),
City nvarchar (50),
District nvarchar (50),
StateProvince nvarchar (50),
Region nvarchar (50),
Country nvarchar (50)
);
CREATE TABLE TimeByDay_2_V39
(
TimeId int PRIMARY KEY,
TheDay nvarchar (15),
Date datetime,
MonthOfYear nvarchar (8),
WeekOfYear nvarchar (8),
Week smallint,
Year nvarchar (4),
TheMonth nvarchar (15),
Month smallint,
QuarterOfYear nvarchar (8),
Quarter nvarchar (2)
);
CREATE TABLE Warehouse_3_V39
(
WarehouseId int PRIMARY KEY,
WarehouseName nvarchar (100),
Owner nvarchar (50),
WPlace nvarchar (255),
WarehouseClass int,
WDescription nvarchar (255)
);
CREATE TABLE Product_4_V39
(
ProductId int PRIMARY KEY,
ProductName nvarchar (255),
Brand nvarchar (50),
ProductClass int,
RetailPrice money,
StockNumber float,
ProductSubCategory nvarchar (15),
ProductCategory nvarchar (15),
ProductDepartment nvarchar (15),
```

```
    ProductFamily nvarchar (15)
    );
    CREATE TABLE Customer_5_V39
    (
    CustomerId int PRIMARY KEY,
    FirstName nvarchar (100),
    LastName nvarchar (100),
    Gender nvarchar (1),
    IncomeAmount money,
    Marriage nvarchar (1),
    Place nvarchar (100)
    );
    CREATE TABLE Promotion_6_V39
    (
    PromotionId int PRIMARY KEY,
    PromotionName nvarchar (100),
    Amount money,
    EffectiveDate datetime,
    EndedDate datetime,
    MediaType nvarchar (255)
    );
    CREATE TA BLE Fact_0_V39
    (
    ExpenseAmount float,
    AccountId int REFERENCES Account_0_V39(AccountId),
    StoreId int REFERENCES Store_1_V39(StoreId),
    TimeId int REFERENCES TimeByDay_2_V39(TimeId),
    CONSTRAINT Fact_0_V39_Key PRIMARY KEY (AccountId, StoreId, TimeId)
    );
    CREATE TABLE Fact_1_V39
    (
    InventoryAmount float,
    WarehouseId int REFERENCES Warehouse_3_V39(WarehouseId),
    StoreId int REFERENCES Store_1_V39(StoreId),
    ProductId int REFERENCES Product_4_V39(ProductId),
    TimeId int REFERENCES TimeByDay_2_V39(TimeId),
    CONSTRAINT Fact_1_V39_Key PRIMARY KEY (WarehouseId, StoreId, ProductId, TimeId)
    );
    CREATE TABLE Fact_2_V39
    (
    SalesAmount float,
    CustomerId int REFERENCES Customer_5_V39(CustomerId),
    ProductId int REFERENCES Product_4_V39(ProductId),
    PromotionId int REFERENCES Promotion_6_V39(PromotionId),
    StoreId int REFERENCES Store_1_V39(StoreId),
    TimeId int REFERENCES TimeByDay_2_V39(TimeId),
    CONSTRAINT Fact_2_V39_Key PRIMARY KEY (CustomerId, ProductId, PromotionId,
StoreId, TimeId)
    );
```

Figure 5.8 An SQL Script in DDL for the Warehouse corresponding to the Example in Figures 5.6 & 5.7

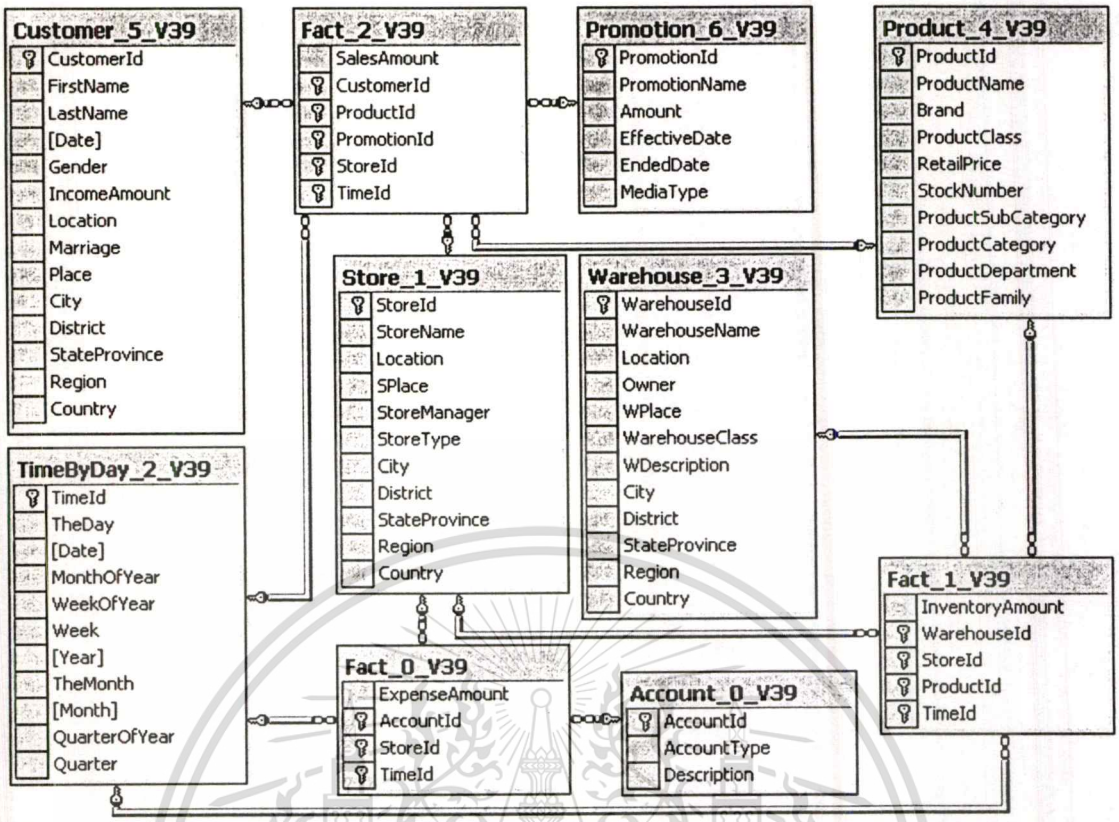


Figure 5.9 The Output of the FoodMart 2000 Warehouse Using MS SQL Server 2000

5.5 Summary

An initiative approach to warehouse schema design based on the NIAM modeling with a deep-structured natural language interface has been presented so that the user effort to be put into the whole of data warehouse design is reduced. In addition, the work is able to be concatenated with the data extraction, transformation and loading (ETL) process from operational sources which are formerly designed using the NIAM modeling. The warehouse schema output is the SQL script in DDL from which a warehouse database output is easily generated using some available DBMS. And these table structures, fact structures and dimensions, in the output database are well prepared in the form of cubes for the ETL process to store data from operational sources to be further processed by several existing front-end tools in order to make reports for analysis, prediction and decision support.

Chapter 6

The Inter-Version Transformation

For our aim at a temporal information warehouse with schema versioning, time aspects are used to define a life span of each warehouse structure version (SV), the validity of all structural elements within an SV, and the relationships among SVs from evolutions. When an SV is evolved from a current one of a temporal warehouse due to the change at the schema level, the evolution leads to a new fact dependent on the source fact which changed so that a new version is created for tracking the change with all elements in the evolved version plus elements after the change.

Accordingly, this chapter focuses on only the evolutions which need the transformations between source and target versions in order to support users to analyze data with no care about structural changes. As an extension, transforming between two versions which does not have direct relationships with each other is easily derived. In addition to the purpose of this chapter, how to manage warehouse SVs, and how to set up relationships among warehouse SVs are also discussed.

6.1 Taxonomy of Warehouse Schema Evolutions

Since schema evolutions may lead to a new fact which is independent of other facts in the current version, or a new fact which is dependent on a source fact and perhaps on a dimension in the current version, this section briefly reviews a few kinds of schema evolutions so that some of them will be selected for the work according to the inter-version transformation need. Given a current structure version SVi, all its elements are fact structures with their composite keys and their measure attributes; and dimensions with their dimension levels organized in terms of hierarchies and descriptive attributes. Dimension keys are regarded as dimension levels, too. This section will give an overall view on warehouse schema evolutions. Hence, [32, 20, and 14] may be referred for further discussion.

6.1.1 Rename an element of an SV

The evolution will occur when a user would like to rename an object of their interest, e.g., a dimension, a dimension level, a measure attribute, or a fact. Only the name visible for the user is going to be changed, but the content kept in the warehouse is intact. Therefore, our work ignores this case for the simplicity.

6.1.2 Add/Remove a measure attribute into/from a fact structure Fi

Adding a measure attribute of interest means that a new fact structure Fi+1 is created independently. That is there is no transformation between the new version SVi+1 which includes the new fact structure Fi+1 and the current one SVi. In contrast, removing a measure attribute will lead to the removal of the fact structure Fi composed of the measure attribute in SVi. The version SVi+1 obtained is the same as SVi without the fact structure Fi. Because of the non-existence of transformation, the evolution is not examined.

6.1.3 Add a dimension Da into the combination of dimensions in a fact structure Fi

Let Fi be formally represented as (DKey1, DKey2, ..., DKeyk, M) where a composition of DKey1, .., and DKeyk is the key of Fi and they refer to the key levels of dimensions D1, ..., Dk, respectively, in SVi; M is a measure attribute of Fi. Given Da, a new dimension added into the combination of dimensions in the fact structure Fi. All components including the key DKeya, dimension levels, hierarchies, and user-defined attributes for descriptive purposes are specified by users. Importantly, users also define the effective starting time for the evolution. This point in time becomes the valid starting point in time for the new version SVi+1 and the valid end point in time for SVi.

The addition of a dimension leads to a new fact structure Fi in a new version SVi+1 where

- SVi+1 inherits all features and properties from SVi except for the evolved fact structure Fi.
- SVi+1 has a new fact structure Fi. This makes SVi+1 different from SVi.
- Fi in SVi+1 has the same measure attribute M as Fi in SVi, but different relationships with dimensions: Fi = (DKey1, DKey2,..., DKeyk, DKeya, M).

The transformation from SVi to SVi+1 is required only for Fi in SVi and transformed measure data in Fi are kept in Fi in SVi+1. For the opposite direction, there is no transformation from SVi+1 to SVi because data in all facts different from Fi in SVi+1 are the same as one in SVi, and data in Fi in SVi+1 are normally transferred to Fi in SVi using the slice operation.

6.1.4 Remove a dimension Da from the combination of dimensions in a fact structure Fi

Let Fi be formally represented as (DKey1, DKey2, ...DKeya, ..., DKeyk, M) where a composition of DKey1, .., DKeya, ..., and DKeyk is the key of Fi and they refer to the key levels of dimensions D1, ..., Da, ..., Dk, respectively, in SVi; M is a measure attribute of Fi. Da for some a in [1...k] is the dimension identified by DKeya and Da is going to be removed from the combination of dimensions in the fact structure Fi in SVi.

If Da is shared by other fact structures, then Da is still kept in the warehouse. Otherwise, Da will be completely removed from the warehouse up to the user choice of the existence of Da. A new version SVi+1 is created with all fact structures different from Fi in SVi and the addition of the fact structure Fi where Fi is changed to (DKey1, ..., DKeyk, M) without DKeya. Users also define the effective starting time for the evolution. This point in time becomes the valid starting point in time for the new version SVi+1 and the valid end point in time for SVi.

The transformation from SVi+1 to SVi is required only for Fi in SVi+1 and transformed measure data are kept in Fi in SVi. For the opposite direction, there is no transformation from SVi to SVi+1 because data in all fact structures different from Fi in SVi are the same as one in SVi+1, and data in Fi in SVi is normally transferred to Fi in SVi+1 using the slice operation.

6.1.5 Add a dimension level DLn+1 into a hierarchy so that DLn+1 is at the level lower than a dimension level DLn in a dimension Da

The addition of a dimension level into a dimension Da in SVi will affect the granularity of fact structures associated with Da if and only if the dimension level is the level following the key of the dimension, that is, if the key is considered as a special level in any hierarchy in the dimension Da, the dimension level which is of interest is the

lowest one in all hierarchies that it contributes. Other dimension levels different from the lowest one have no impact on the granularity of the fact. This leads to no impact on inter-version transformations although the change is a schema evolution. For this reason, we do not pay attention to such an evolution.

Inter-version transformations are required to perform on SVi to transform measure data of all affected fact structures belonging to SVi into SVi+1 which will be a current version of the temporal information warehouse.

6.1.6 Remove a dimension level DLn+1 from a dimension Da

Similar to the addition of a dimension level, the removal of a dimension level in a Dimension Da associated with fact structures Fi will affect the granularity of Fi if and only if the dimension level is the level following the key of the dimension, that is, if the key is considered as a special level in any hierarchy in the dimension Da, the dimension level which is of interest is the lowest one in all hierarchies that it contributes. Other dimension levels different from the lowest one have no impact on the granularity of Fi. So, no impact on inter-version transformations occurs although the change is a schema evolution. Also, we do not pay attention to such an evolution.

Opposite to the previous kind of evolutions, inter-version transformations are required to perform on SVi+1 to transform measure data of all influenced fact structures belonging to SVi+1 into SVi.

6.2 The Design of Warehouse Schema Evolutions

6.2.1 The Sub-Architecture for Schema Versioning in the TIWA1 Architecture

Even though a number of schema evolutions may emerge in a warehouse system and each respective structural change leads to a new SV for the latest environmental conditions, not all of them require measure data transformations between a new version and an evolved current version. From this point, as shown in the taxonomy of warehouse schema evolutions, our inter-version transformations are going to be applied on several crucial situations that ask for the representation of measure data in an SV using the structure of another SV. Such crucial situations to be tackled are adding/removing a dimension, and adding/removing a dimension level.

The sub-architecture for schema versioning in the TIWA1 architecture introduced in Figure 6.1 explains how our system allows users to evolve their warehouse schema of a current SV at the conceptual level through a simple natural language interface. This approach to schema versioning will inherit the efforts from the previous chapter.

We can reuse the proposed schema transformation from a NIAM/ORM conceptual schema into a warehouse schema in the chapter 5 to identify what warehouse schema

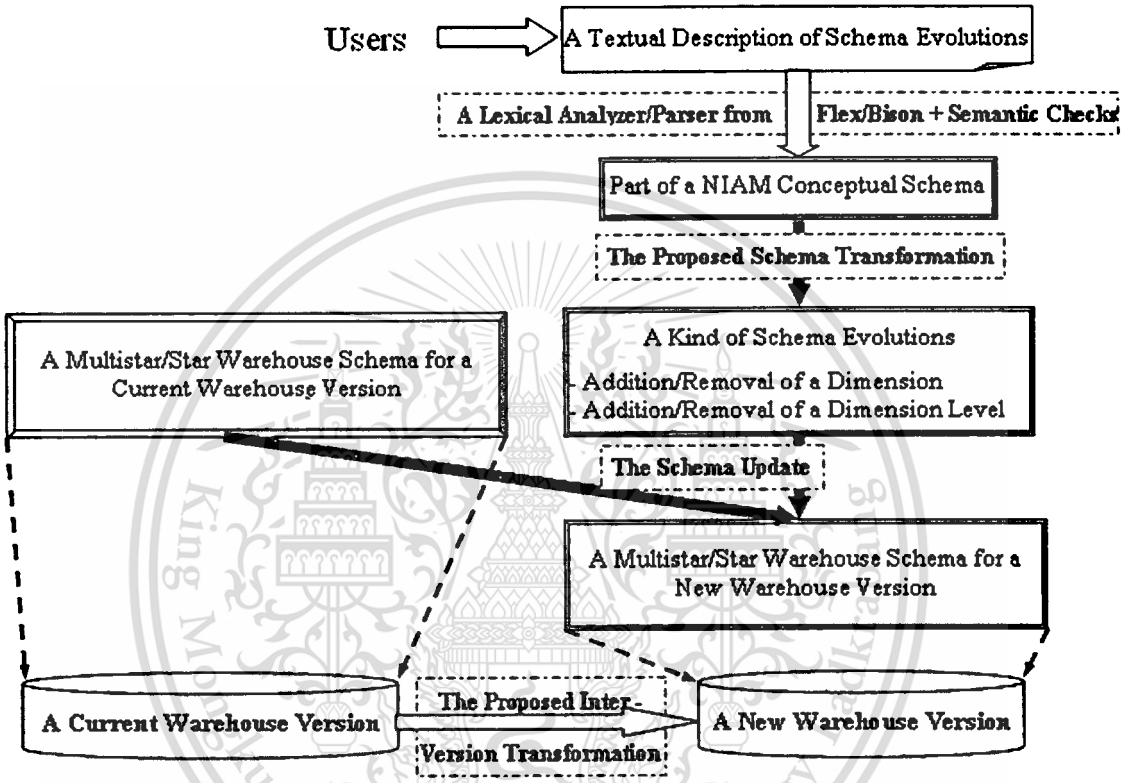


Figure 6.1 The Sub-Architecture in the Proposed Architecture TIWA1 for Schema Versioning

evolutions users are considering. Then it is trivial to obtain a multistar/star warehouse schema corresponding to a new warehouse version from a multistar/star warehouse schema corresponding to a current warehouse version and schema evolutions; and to establish a time-related relationship between these two versions.

With two versions and their relationship, the inter-version transforming process is invoked for the representation of measure data belonging to one version in the structure of the other version. This fact implies that structural changes at the schema level are propagated to data changes at the instance level. Also from the sub-architecture, our

work allows users to do schema versioning with only a chain of versions along the axis Time.

6.2.2 A NIAM/ORM Grammar for the Specification of Warehouse Schema Evolutions

The NIAM/ORM grammar for the specification of warehouse schema evolutions is context-free in the Backus-Naur form (BNF) as presented in Figure 6.2. uXXX are terminal symbols from user inputs, <xxx> are non-terminal symbols. XXX and xxx are reserved terminal symbols which are not allowed to be used as names of object types or rols. All reserved terminal symbols are case-sensitive. These kinds of symbols carry the same meanings as ones in the section 5.4 of the chapter 5.

```
<nschema> ::= <add_dim> <facts> <constr> <dtype>
        | <add_dl> <facts> <constr> <dtype> <removal>
        | <rem_dim>
        | <rem_dl>
<add_dim> ::= ADDITION INTO uNAME. uNAME EFFECTIVE FROM <efftime> . MODIFIED AS
<naryfact> . END .
<add_dl> ::= ADDITION INTO uNAME. uNAME EFFECTIVE FROM <efftime> . END .
<removal> ::= /*empty*/ | REMOVAL <taryfacts> END .
<rem_dim> ::= REMOVAL FROM uNAME. uNAME EFFECTIVE FROM <efftime> . MODIFIED AS
<naryfact> . END .
<rem_dl> ::= REMOVAL FROM uNAME. uNAME EFFECTIVE FROM <efftime> . <taryfacts> END .
<efftime> ::= uNUMBER / uNUMBER / uNUMBER
<naryfact> ::= uNAME uROLENAME uNAME uROLENAME uNAME | <naryfact> uROLENAME uNAME
<facts> ::= <taryfacts> | <taryfacts> <facts>
<taryfacts> ::= uNAME uROLENAME / uROLENAME uNAME. | uNAME : uNAME uROLENAME /
uROLENAME uNAME.
<constr> ::= CONSTRAINT <identifier> <unique_optional> <entity_section> <subtype_section>
<frequency_section> <equal_section> <exclusive_section> <subset_section> END .
<identifier> ::= uNAME IS UNIQUELY IDENTIFIED BY uNAME : <datatype> ( uNUMBER ) .
        | uNAME IS UNIQUELY IDENTIFIED BY uNAME : <datatype> ( uNUMBER ) . <identifier>
<unique_optional> ::= EACH uNAME uROLENAME <quantity> uNAME.
        | EACH uNAME uROLENAME <quantity> uNAME . <unique_optional>
<quantity> ::= EXACTLY ONE | ZERO OR ONE | ZERO OR MORE | ONE OR MORE
<entity_section>: /*empty*/ | ENTITY <entity>
<entity> ::= uNAME { <enumeratedset> } . | uNAME { <enumeratedset> } . <entity>
        | uNAME [ <continuousrange> ] . | uNAME [ <continuousrange> ] . <entity>
<enumeratedset> ::= uSTRING , uSTRING  | uNUMBER , uNUMBER
        | uSTRING , <enumeratedset> | uNUMBER , <enumeratedset>
<continuousrange> ::= uNUMBER . . uNUMBER
<subtype_section> ::= /*empty*/ | SUBTYPE <subtype>
<subtype> ::= uNAME = <category> . | uNAME = <category> . <subtype>
<category> ::= uNAME uROLENAME uNAME rWITH uNAME <comparison> uSTRING
        | uNAME uROLENAME uNAME rWITH uNAME <comparison> uNUMBER
        | UNION ( <category> , <category> )
        | INTERSECT ( <category> , <category> )
        | MINUS ( <category> , <category> )
<comparison> ::= > | < | >= | <= | = | <>
<frequency_section> ::= /*empty*/ | FREQUENCY <frequency>
<frequency> ::= uNAME : ( <objectname> ) OCCURS uNUMBER TIMES .
        | uNAME : ( <objectname> ) OCCURS uNUMBER TIMES . <frequency>
```

```
        | uNAME:(<objectname>) OCCURS AT LEAST uNUMBER AND AT MOST uNUMBER TIMES .
        | uNAME:(<objectname>) OCCURS AT LEAST uNUMBER AND AT MOST uNUMBER TIMES .
<frequency>
<objectname> ::= uNAME | uNAME, <objectname>
<equal_section> ::= /*empty*/ | EQUALITY <equal>
<equal> ::= FOR EACH uNAME : uROLENAME( uNAME ) IFF uROLENAME( uNAME ) .
        | FOR EACH uNAME : uROLENAME( uNAME ) IFF uROLENAME( uNAME ) . <equal>
<exclusive_section> ::= /*empty*/ | EXCLUSIVITY <exclusive>
<exclusive> ::= NO uNAME PLAYS BOTH uROLENAME( uNAME ) AND uROLENAME( uNAME ) .
        | NO uNAME PLAYS BOTH uROLENAME ( uNAME ) AND uROLENAME( uNAME ) .
<exclusive>
<subset_section> ::= /*empty*/ | SUBSET <subset>
<subset> ::= FOR EACH uNAME : IF uROLENAME( uNAME ) THEN uROLENAME( uNAME ) .
        | FOR EACH uNAME : IF uROLENAME( uNAME ) THEN uROLENAME( uNAME ) . <subset>
<dtype> ::= /*empty*/ | DATATYPE <declaration> END .
<declaration> ::= uNAME : <datatype> ( uNUMBER ) . | uNAME : <datatype> ( uNUMBER ) .
<declaration>
<datatype> ::= bigint | binary | char | datetime | decimal | float | int | money | nchar | numeric | nvarchar |
real | smallint | varchar
```

Figure 6.2 The NIAM/ORM Grammar for the Specification of Warehouse Schema Evolutions

From the grammar above, the textual description of warehouse schema evolutions describes one of four evolution kinds: the addition/removal of a dimension, the addition/removal of a dimension level. In order to explain the grammar more clearly and instruct users how to form their warehouse schema evolutions at the conceptual level using a deep-structured natural language, examples are given for each evolution kind as follows. Let's assume that the first SV1 was obtained from the design of a warehouse that has been presented in the chapter 5. Figure 6.3 shows a NIAM/ORM conceptual schema that models the SV1, and Figure 6.4 shows a corresponding multistar warehouse schema of the SV1 at the logical level.

6.2.2.1 Add a dimension level

The following textual description in Figure 6.5 is for the addition of the dimension level StoreSubType into the dimension Store_1_V44 in the SV1 to obtain an SV2. This level will be added into the hierarchy Store(StoreId) → Store_Type (StoreType) so that a new hierarchy Store(StoreId) → StoreSubType(StoreSubTypeId) → Store_Type(StoreType) is created. Together with a new dimension level StoreSubType, descriptive attributes associated with the new dimension level are included in the schema via binary fact types and/or reference types.

The first part of the textual description contains the name of the conceptual schema modeling the current version and the name of the affected dimension. This part also

includes the effective point in time for a new version. This point in time will be used to end the validity of the current version.

In the second part right after "ADDITION ... END.", simple English sentences are

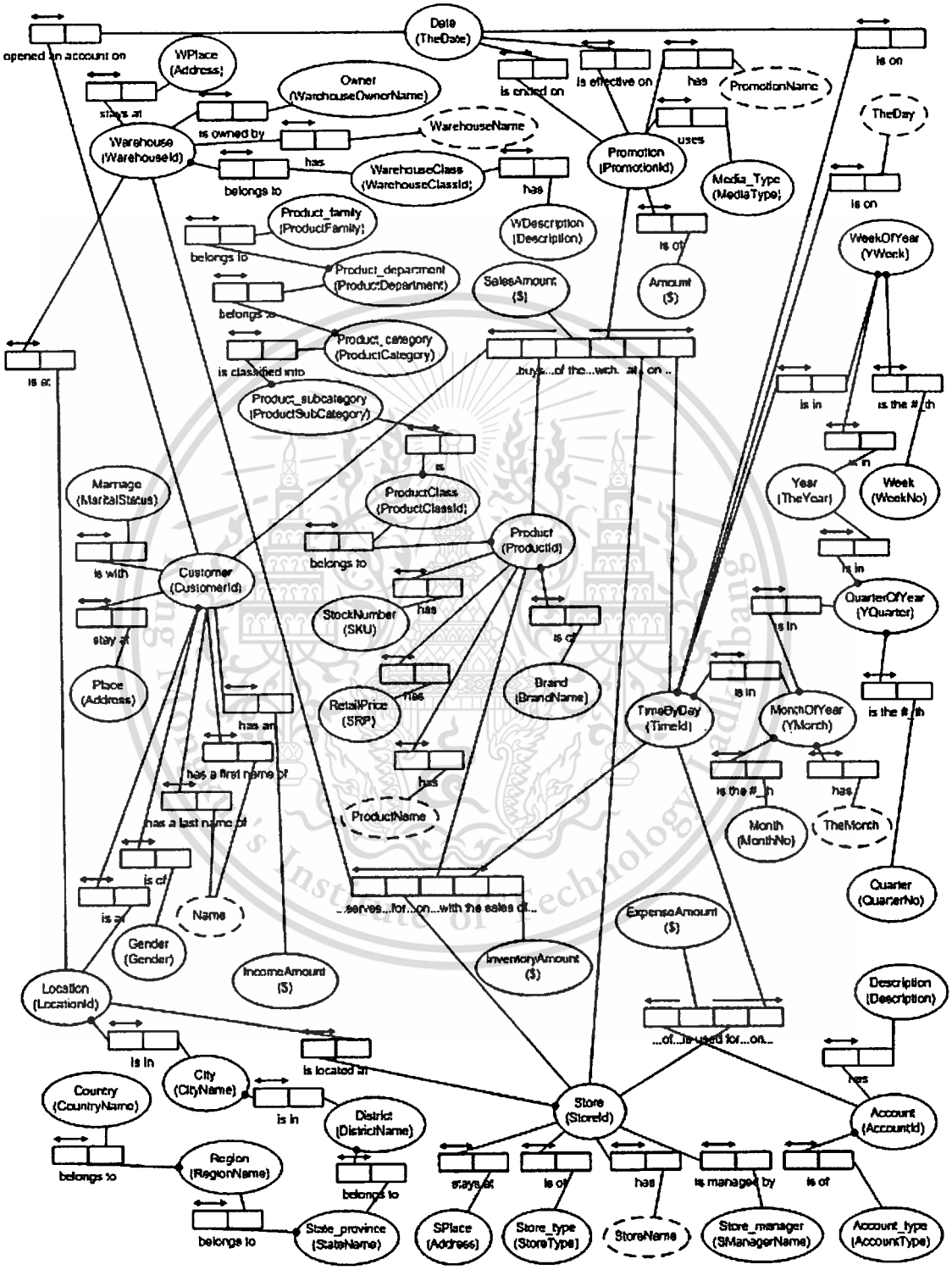


Figure 6.3 A NIAM/ORM Conceptual Schema for the SV1

input for only binary fact types and reference types that express relationships between two objects in the UoD.

The next two parts are for constraint declarations and data type declarations.

The part "REMOVAL ... END." asks users to specify which binary fact types are out of date. It means that relationships between two objects will no exist longer in reality.

The information derived from the textual description along with the information of the conceptual schema of the current version is used to construct a new warehouse schema of a respective version. The output in Figure 6.6 does not show new/old hierarchies in

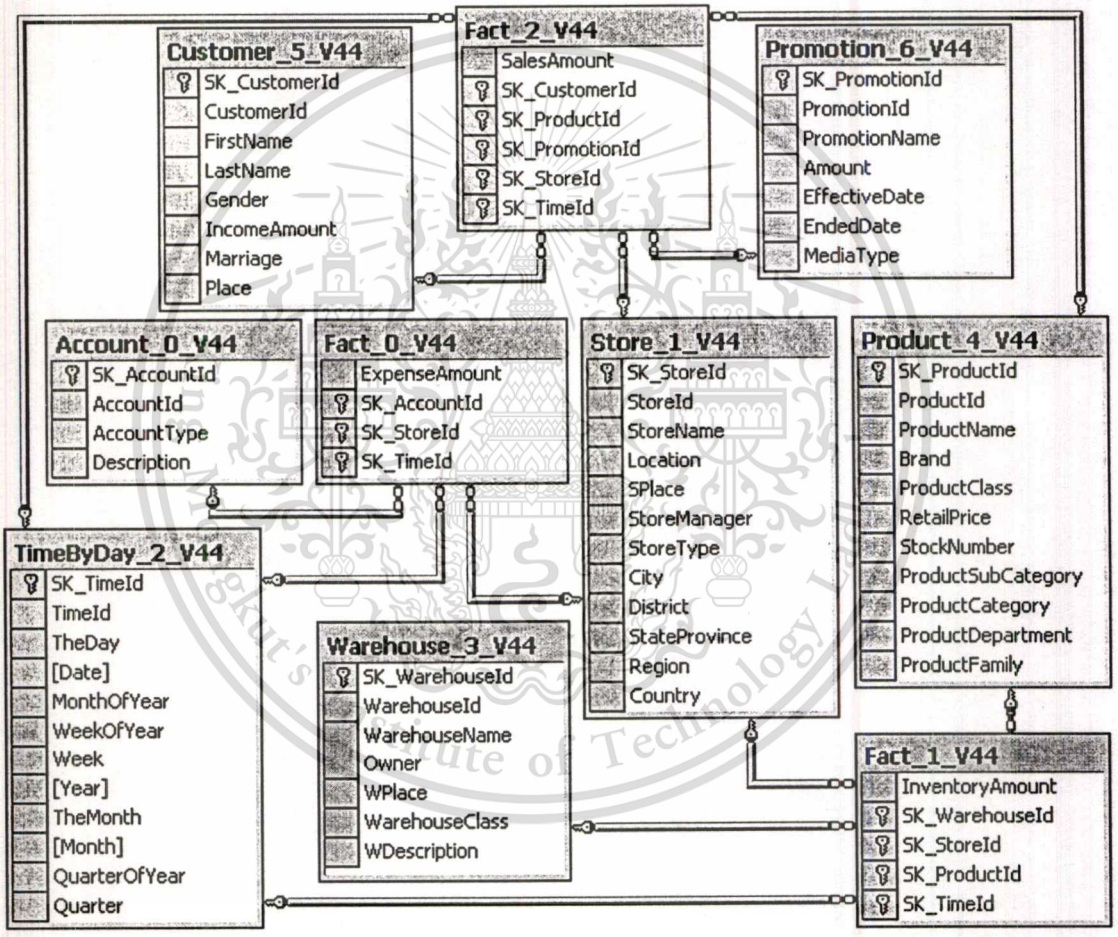


Figure 6.4 The Multistar Warehouse Schema of the SV1

```
ADDITION INTO CSFoodMart5.Store
        EFFECTIVE FROM 02/15/2005.
END.

Store belongs_to/has StoreSubType.
StoreSubType has/is_for StoreSubTypeName.
StoreSubType is_categorized_by/includes Store_Type.
```

```
CONSTRAINT
        StoreSubType IS UNIQUELY IDENTIFIED BY StoreSubTypeId: int (4).
        Store IS UNIQUELY IDENTIFIED BY StoreId: int(4).
        Store_Type IS UNIQUELY IDENTIFIED BY StoreType: nvarchar(255).

        EACH Store belongs_to ZERO OR ONE StoreSubType.
        EACH StoreSubType has ZERO OR MORE Store.
        EACH StoreSubType has EXACTLY ONE StoreSubTypeName.
        EACH StoreSubTypeName is_for ZERO OR MORE StoreSubType.
        EACH StoreSubType is_categorized_by EXACTLY ONE Store_Type.
        EACH Store_Type includes ONE OR MORE StoreSubType.
        EACH Store is_of EXACTLY ONE Store_Type.
        EACH Store_Type has ZERO OR MORE Store.
END.

DATATYPE
        StoreSubTypeName: nvarchar(100).
END.

REMOVAL
        Store is_of/has Store_Type.
END.
```

Figure 6.5 A Textual Description for the Addition of a Dimension Level StoreSubType into the Dimension Store_1_V44

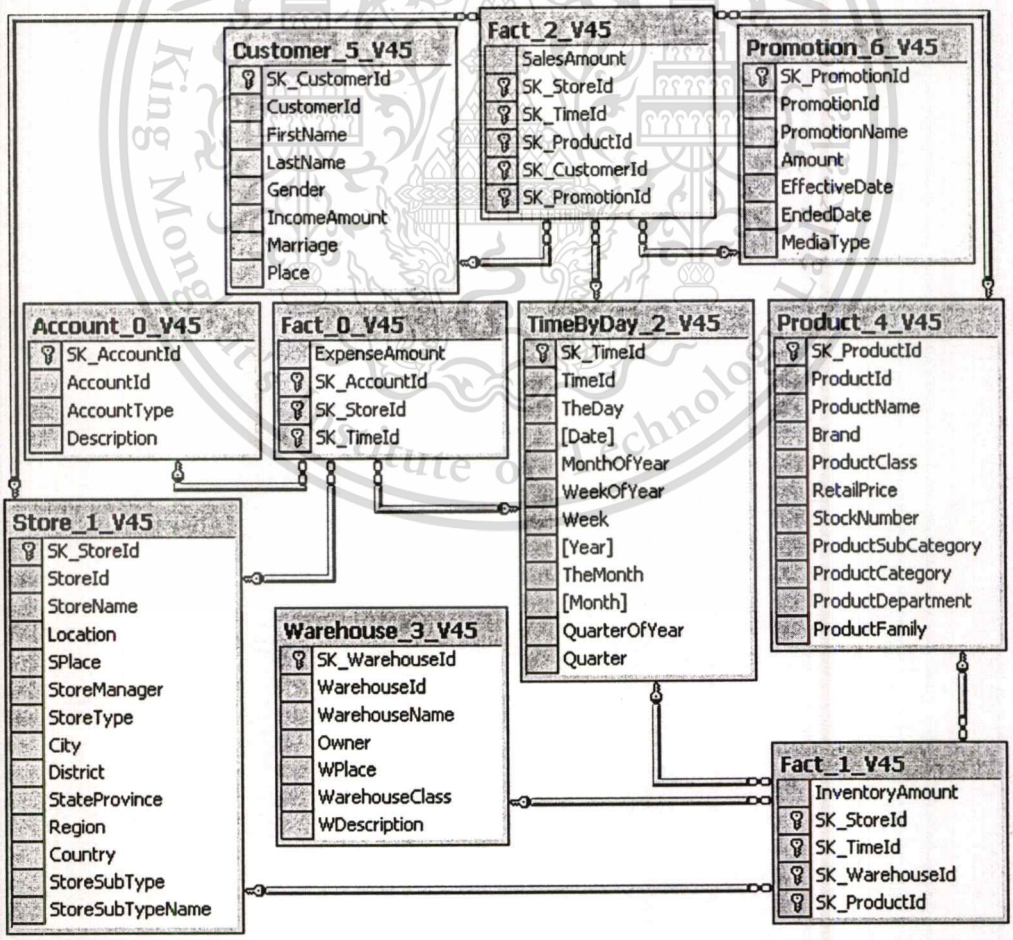


Figure 6.6 The Multistar Warehouse Schema of the SV2

dimensions. However, the information about fact structures, dimensions, levels, their descriptive attributes and their hierarchies is kept in the fifth normal form (5NF) meta tables obtained from the information warehouse conceptual meta schema in Figure 5.3.

The addition of the dimension level StoreSubType will affect all fact structures that have relationships with the dimension Store_1_V44 in SV1.

6.2.2.2 Remove a dimension level

The removal of a dimension level is rare in practice because every thing evolves over time and organizational business often expands. Consequently, information is required more and more in details. Nevertheless, it might occur in some enterprise. A description in Figure 6.7 shows how users represent their need of removing a dimension level using our defined natural language.

```
REMOVAL FROM ConceptualSchemaName.EntityTypeName
/*EntityTypeName corresponds to a name of an object type that takes part in at least one n-ary fact type for n>2*/
        EFFECTIVE FROM 12/3/2000.
        Time is_of/has Month.     /*a binary fact type that will be removed from the conceptual schema*/
END.
```

Figure 6.7 A Textual Description for the Removal of a Dimension Level

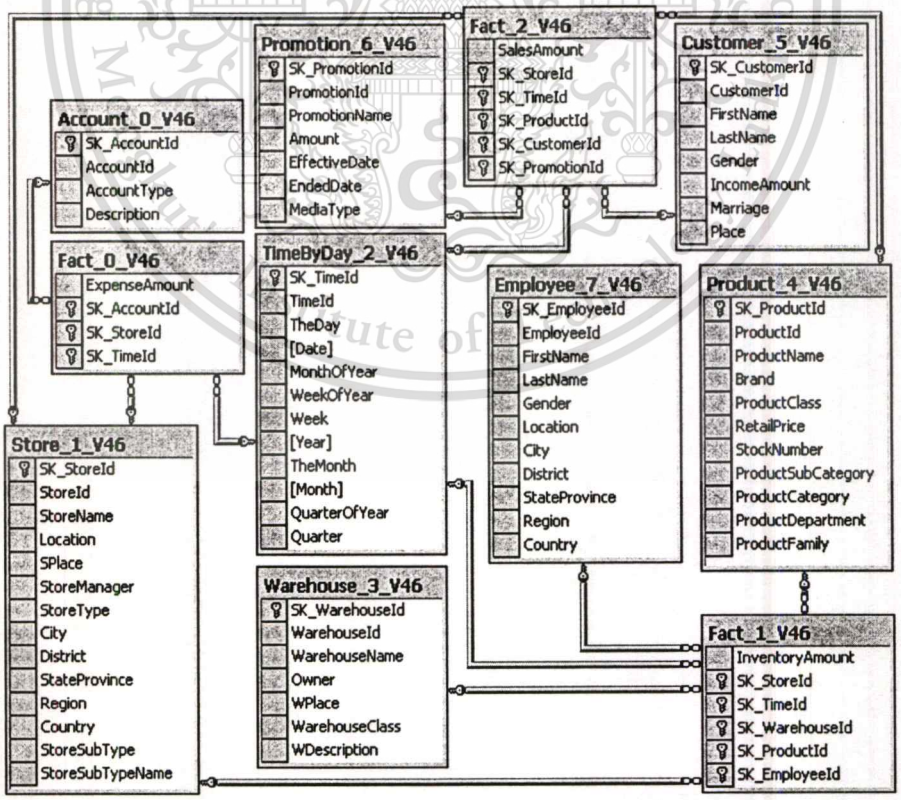


Figure 6.8 The MultiStar Warehouse Schema of the SV3

6.2.2.3 Add a dimension

The following textual description in Figure 6.9 is used for the addition of a dimension Employee_7_V46 into the fact structure Fact_1_V45 in the SV2 to obtain an SV3. This fact structure Fact_1_V45 corresponds to the 5-ary fact type InventorySales in the conceptual schema that describes the SV2. All hierarchies among dimension levels and descriptive attributes of dimension levels in the new dimension are represented via binary fact types and reference types. These binary fact types and reference types are input using simple English sentences. Besides, users have to redefine the 5-ary fact type InventorySales with the occurrence of the new object type Employee at the conceptual level that corresponds to the new dimension Employee_7_V46 at the logical level. All constraint declarations and data type declarations are also included in the description. The output is a new warehouse schema for a new version SV3 in Figure 6.8.

```
ADDITION INTO CSFoodMart5.InventorySales
        EFFECTIVE FROM 06/30/2006.
        MODIFIED AS Warehouse serves Store for Product on TimeByDay with_the_sales_of
InventoryAmount with_responsibility_of Employee.
END.

Employee has_a_first_name_of/belongs_to FirstName.
Employee has_a_last_name_of/belongs_to LastName.
Employee is_of/has Gender.
Employee lives_at/is_with Location.
Location is_in/has City.
City is_in/has District.
District belongs_to/has StateProvince.
Region belongs_to/has Country.
StateProvince belongs_to/has Region.

CONSTRAINT
        Employee IS UNIQUELY IDENTIFIED BY EmployeeId : int(4).
        Location IS UNIQUELY IDENTIFIED BY LocationId : int(4).
        City IS UNIQUELY IDENTIFIED BY CityName : nvarchar(50).
        District IS UNIQUELY IDENTIFIED BY DistrictName : nvarchar(50).
        Region IS UNIQUELY IDENTIFIED BY RegionName : nvarchar(50).
        Country IS UNIQUELY IDENTIFIED BY CountryName : nvarchar(50).
        StateProvince IS UNIQUELY IDENTIFIED BY StateName : nvarchar(50).
        Gender IS UNIQUELY IDENTIFIED BY GenderCode : nvarchar(1).
        Warehouse IS UNIQUELY IDENTIFIED BY WarehouseId : int(4).
        Store IS UNIQUELY IDENTIFIED BY StoreId : int(4).
        Product IS UNIQUELY IDENTIFIED BY ProductId : int(4).
        TimeByDay IS UNIQUELY IDENTIFIED BY TimeId : int(4).
        InventoryAmount IS UNIQUELY IDENTIFIED BY AInventory : float(8).

        EACH Region belongs_to EXACTLY ONE Country.
        EACH Country has ONE OR MORE Region.
        EACH StateProvince belongs_to EXACTLY ONE Region.
        EACH Region has ONE OR MORE StateProvince.
        EACH District belongs_to EXACTLY ONE StateProvince.
        EACH StateProvince has ONE OR MORE District.
```

```
        EACH Location is_in EXACTLY ONE City.
        EACH City has  ONE OR MORE Location.
        EACH City is_in EXACTLY ONE District.
        EACH District has ONE OR MORE City.
        EACH Employee lives_at EXACTLY ONE Location.
        EACH Location is_with  ZERO OR MORE Employee.
        EACH Employee has_a_first_name_of EXACTLY ONE FirstName.
        EACH FirstName belongs_to ZERO OR MORE  Employee.
        EACH Employee has_a_last_name_of EXACTLY ONE LastName.
        EACH LastName belongs_to ZERO OR MORE  Employee.
        EACH Employee is_of EXACTLY ONE Gender.
        EACH Gender has ONE OR MORE  Employee.

        ENTITY
        Gender {'M', 'F'}.
END.

DATATYPE
        FirstName : nvarchar(100).
        LastName : nvarchar(100).
END.
```

Figure 6.9 A Textual Description for the Addition of the Dimension Employee_7_V46

6.2.2.4 Remove a dimension

A description in Figure 6.7 shows how users represent their need of removing a dimension from the association with a fact structure using our defined language. Users have to redefine the n-ary fact type for n>2 without the occurrence of the object type that corresponds to a dimension at the logical level.

```
REMOVAL FROM ConceptualSchemaName.CompoundObjectName
/* CompoundObjectName corresponds to a name of an n-ary fact type for n>2 */
        EFFECTIVE FROM 12/3/2000.
        MODIFIED AS Customer buys Product at Store on Time.
END.
```

Figure 6.10 A Textual Description for the removal of a dimension

6.3 The Inter-Version Transformation Techniques

Basically, the nature of data warehouse is related to time along with data spanning over periods of time. During a long time interval, it is possible for the underlying structure of a data warehouse to be changed to conform itself to the latest environmental conditions. If different structures of a temporal data warehouse are allowed to be seen by users, it is a challenge to present the time varying data that comply with the particular schema chosen by the users. Making the most of a well known field - data mining, this section presents the development of inter-version transformation in temporal data warehouses using the proportion notion together with the K-means clustering technique.

The result from our technique is found to be more accurate than one from existing techniques. In addition, this chapter allows users to update the warehouse structural changes at the conceptual level using a deep-structured natural language interface. Once the system captures these structural changes in meta tables, inter-version transformations between a current structure version and a new one are carried out automatically using our proposed technique. The technique uses data in the structure of the target version to calculate probabilities, known as their factors, for schema transformations of the source structure version. With such an action, warehouse data in the source version mapped into the target version have the same characteristics or data distribution as ones in the target version. Moreover, our technique makes transformed data more truthful and consistent with the structure of the target version.

6.3.1 Classical Techniques for the Inter-Version Transformation

In respect with automatically transforming measure data among versions in temporal warehousing, we are going to take [32] and [20] into account with uniformity transformations as earlier mentioned in the chapter 1.

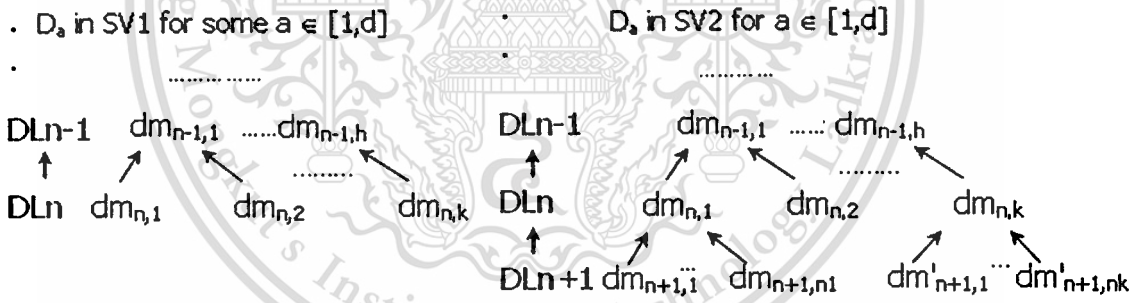


Figure 6.11 The Dimension D_a in SV1 and SV2

Given a temporal data warehouse schema TDW, let D_i for i = $\overline{1,d}$ be dimension schemas, and F be the fact schema with complete measure data for all the combinations of keys of the dimension tables. Let TDW = {SV1, SV2}, where an SV stands for a structure version of the temporal data warehouse TDW, and SV2 is obtained from SV1 by adding the dimension level DL_{n+1}, the lowest level in the dimension schema D_a, for a in [1, d]. That is, SV2 has D_a with the lowest dimension level DL_{n+1}, and SV1 has D_a with the lowest dimension level DL_n which is the parent level (upper level) of the

dimension level DL_{n+1} as shown in Figure 6.11. Within the above described context, we need to transform measure data of F in SV1 into the new structure in SV2.

Existing techniques using weighting/confidence factors w make a division of measure data related to dimension members at the dimension level DL_n in SV1 equally for every dimension member at the dimension level DL_{n+1} in SV2 [32], [22], [20]. Based on this simple technique, the output sometimes becomes unacceptable and impractical. For example, within a temporal data warehouse, the fact table in SV_{r+1} contains monthly measure data about the number of sold sweaters and the fact table in SV_r contains the quarterly measure data. Consider the fourth quarter including October, November, and December in 2002 in SV_{r+1} and the fourth quarter in 2001 in SV_r. Now transform the measure data of 300 sweaters corresponding to the fourth quarter in 2001 into the structure of SV_{r+1}. We obtain 100 sweaters from a division of 300 by 3 for each month in the fourth quarter in 2001 in SV_{r+1}. It is not truthful because the number of sold sweaters in December is much more than those in October and November every year. For their methods applying on transformations from SV1 to SV2 in the TDW, weighting factors w are below calculated [32], [22], and [20]:

For each $dm_{n+1,t}$ where $t = \overline{1,k}$, $w_1 = w_2 = \ldots = w_{n_t} = 1/n_t$

Where k is the total number of possible dimension members $dm_{n,t}$ for $t = \overline{1,k}$ at the level DL_n in the dimension table D_a, and n_t is the total number of possible dimension members $dm_{n+1,j}$ for $j = \overline{1,n_t}$ and $t = \overline{1,k}$ at the level DL_{n+1} in dimension table D_a as presented in Figure 6.12.

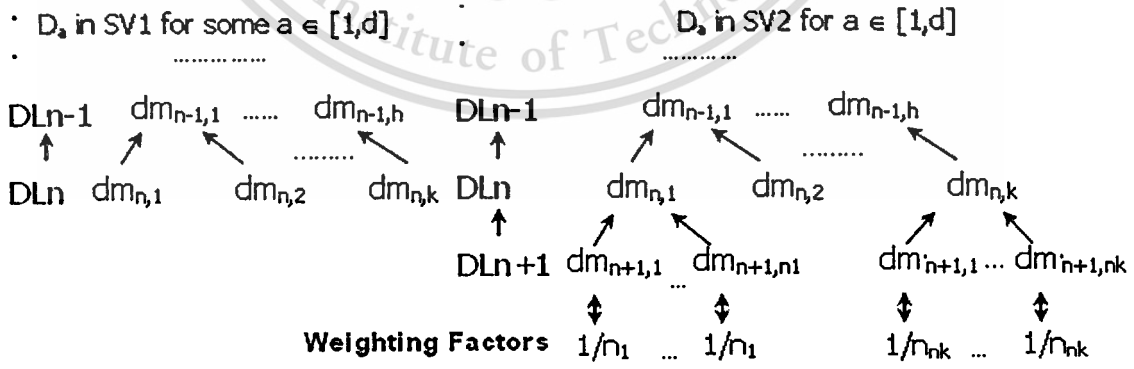


Figure 6.12 Weighting Factors for Dimension Members of the New Dimension Level DLn+1 in SV2

6.3.2 The Proposed Technique for the Inter-Version Transformation

For our proposed inter-version transformation technique, we figure out how much each dimension member at the dimension level DL_{n+1} in SV2 contributes. The proportion of the dimension member at the dimension level DL_{n+1} in SV2 to split measure data related to the parent dimension member at the dimension level DL_n in SV1 is shown in Figure 6.13. Probabilities corresponding to each dimension member $dm_{n,t}$ for $t = \overline{1,k}$ at the dimension level DL_n in SV1 are computed by using measure data in SV2 according to the definition of proportion "Proportion is the ratio of the chances favoring a certain thing happening to all the chances for and against it, or in other words, proportion is the ratio of the number of favorable cases to the total number of all cases". This definition is confined to the condition that all the chances (cases) are equally probable.

Two approaches are investigated to allow users to make a choice of transformation. The first one provides users with probabilities from individual measure data, and the other with probabilities from average measure data. For example, consider a temporal data warehouse with two structure versions SV_r and SV_{r+1}. SV_{r+1} that supports monthly data is valid in two years 2000 and 2001. SV_r that supports quarterly data is valid in 1999. For the first approach, probabilities for transformation from measure data of 1999 in SV_r are calculated by using measure data of 2000 or 2001 in SV_{r+1}. As for the second approach, probabilities for transformation from measure data of 1999 in SV_r are calculated by using average measure data of all measure data of 2000 and 2001 in SV_{r+1}.

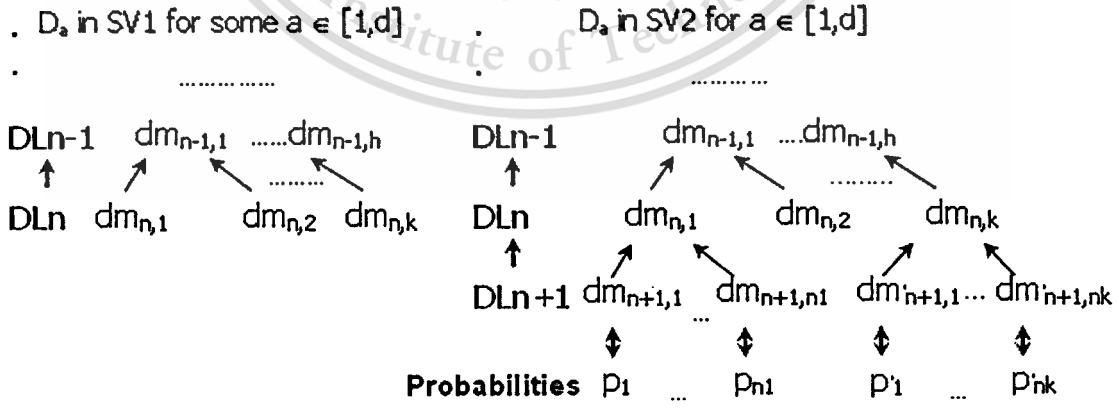


Figure 6.13 Probabilities for Dimension Members of the New Dimension Level DLn+1 in SV2

For the first approach in the TDW, let P_1, P_2, ... P_{nt} (or in other notations, P_j, where j = $\overline{1,n_t}$) be the probabilities for transformations of each dimension member $dm_{n,\,t1}$, where t1 = $\overline{1,k}$ at the dimension level DL_n in SV1 to dimension members $dm_{n+1,j}$, where j = $\overline{1,n_t}$ at the dimension level DL_{n+1} in SV2 using measure data related to dimension members at the dimension level DL_{n+1} in SV2 that have the same parent dimension member $dm_{n,t2}$, where t2 = $\overline{1,k}$ at the dimension level DL_n in SV2 which has the same characteristics as $dm_{n,t1}$ corresponding to each combination of other keys of the other dimension tables in the fact table that is the same in SV1 and SV2. Each P_j is calculated as follows:

$$P_j = \frac{Measure\_Data\_DM_{n+1,j}}{\sum_{z=1}^{n_t} Measure\_Data\_DM_{n+1,z}}, j = \overline{1,n_t} \qquad (6\text{-}1)$$

Where $Measure\_Data\_DM_{n+1,j}$ is a sample measure value associated with each dimension member $dm_{n+1,j}$ for j = $\overline{1,n_t}$ at the dimension level DL_{n+1} in SV2.

As for the second approach in the TDW, let $\overline{P_j}$, where j = $\overline{1,n_t}$ be the probabilities for average transformations of each dimension member $dm_{n,\,t1}$ where t1 = $\overline{1,k}$ at the dimension level DL_n in SV1 from average measure data related to dimension members at the dimension level DL_{n+1} in SV2 that have the same parent dimension member $dm_{n,\,t2}$ where t2 = $\overline{1,k}$ at the dimension level DL_n in SV2. Average measure data for each $dm_{n+1,j}$ where j = $\overline{1,n_t}$ in SV2 is gained over all the valid periods of time of a structure version with all the combinations of other keys of the other dimension tables in the fact table in SV2. The period mentioned is considered as the duration of time in which there are the full combinations of all the keys of all the dimension tables in the fact table of a structure version of a temporal data warehouse. For example, consider a temporal data warehouse with 2 structure versions SV_r and SV_{r+1}. SV_{r+1} is valid in two years 2000 and 2001 and SV_r in one year 1999. Average measure data is computed over two years 2000 and 2001 for transformations from SV_r to SV_{r+1} and each year is one period for the complete combinations of all the keys of all the dimension tables in the fact table in SV_{r+1}. For the TDW, these $\overline{P_j}$, where j = $\overline{1,n_t}$ are computed as follows from average measure data of each dimension member $dm_{n+1,\,j}$ where j = $\overline{1,n_t}$ belonging to $dm_{n,\,t2}$ where t2 = $\overline{1,k}$ in SV2.

$$\overline{P_j} = \frac{\overline{Measure\_Data\_DM_{n+1,j}}}{\sum_{z=1}^{n_t} \overline{Measure\_Data\_DM_{n+1,z}}}, j = \overline{1, n_t} \qquad (6\text{-}2)$$

Where $\overline{Measure\_Data\_DM_{n+1,j}}$ for j = $\overline{1, n_t}$ is the average measure data for each dimension member $dm_{n+1,j}$ for j = $\overline{1, n_t}$.

The problem is how to calculate $\overline{Measure\_Data\_DM_{n+1,j}}$ for j = $\overline{1, n_t}$ using the modified 2-means as briefly presented as follows. This step is pre-processing data attending the computation of $\overline{Measure\_Data\_DM_{n+1,j}}$ for j = $\overline{1, n_t}$ to detect and eliminate noisy data from the group of measure data related to each dimension member $dm_{n+1,j}$ for j = $\overline{1, n_t}$.

Input: The number of desired clusters is 2 and the set of elements is C = {d_1, d_2,..., d_N}.

Output: Two clusters C_1 and C_2

The Modified 2-means algorithm [34]:

Assign initial values for means m_1, m_2 which differ from each other;

Repeat

Assign each item d_i for i = $\overline{1, N}$ to the cluster which has the closest mean;

Calculate new mean for each cluster;

Until m_1, m_2 is not varying;

The time complexity of the modified 2-means is O (2rN) where r is the number of iterations. The modified 2-means finds a local optimum and may actually miss the global optimum [34], [23]. The purpose of the use of the modified 2-means is to separate C into two clusters including one significant cluster C_1 which contains more elements and has a smaller mean than the other, and one non-significant cluster C_2 which contains unexpected elements not complying with the general behavior or the general model of the whole data and having a strong impact on the average value of all elements in C.

Going back to our transformation from SV1 to SV2, assume SV2 has x periods. In one period, the complete data in the fact table in SV2 is the combinations of η key values for each dimension of d dimension tables with i = $\overline{1, d}$ and i ≠ a and n_t dimension members $dm_{n+1,j}$ for j = $\overline{1, n_t}$ at the dimension level DL_{n+1} for each dimension member of k dimension members $dm_{n,t}$ for t = $\overline{1, k}$ at the dimension level DL_n in the dimension table D_a. So, the total number of combinations in the fact table in one period is

$(n_1+..+n_k)*m_1*...*m_d$ or $\sum_{t=1}^{k} n_t * \prod_{i=1}^{d} m_i$ and in x periods is $x*\sum_{t=1}^{k} n_t * \prod_{i=1}^{d} m_i$, where m_i for i = $\overline{1,d}$ and i ≠ a is the number of key values, with respect to the number of rows, in the dimension table D_i.

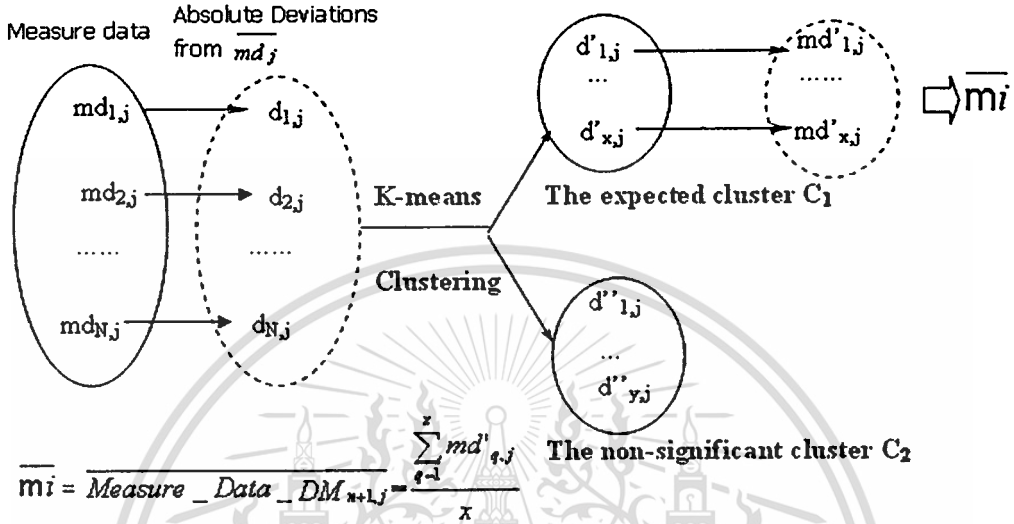


Figure 6.14 The Procedure Using the K-means Clustering Technique for Average Measure Values

For each dimension member $dm_{n+1,j}$ for j = $\overline{1,n_t}$ at the dimension level DL_{n+1} in SV2, we need to calculate the average measure data $\overline{Measure\_Data\_DM_{n+1,j}}$ with all the measure data related to the combinations of other keys of other dimension tables and the keys corresponding to $dm_{n+1,j}$ of the dimension table D_a in the fact table in x periods.

Let these measure data be $md_{h,j}$ for h = $\overline{1, x*\prod_{i=1}^{d} m_i}$ for each $dm_{n+1,j}$. Among these measure data $md_{h,j}$ may there be some elements which have an inconsistent behavior with the others. Therefore, we recognize the non-significant elements using the modified 2-means above with the set of deviations of all the elements from the mean $\overline{md_j}$ of the measure data $md_{h,j}$. Users may apply some formula different from the arithmetic mean to calculate $\overline{md_j}$.

$$N = x*\prod_{i=1}^{d} m_i \qquad (6\text{-}3)$$

$$\overline{md_j} = \frac{1}{N}\sum_{h=1}^{N} md_{h,j} \qquad (6\text{-}4)$$

Deviations of the measure data $md_{h,j}$ from the mean $\overline{md_j}$ are d_{hj} for h = $\overline{1,N}$ computed as follows:

$$d_{hj} = |md_{h,j} - \overline{md_j}| \qquad (6\text{-}5)$$

The larger the deviation of $md_{h,j}$, the further md_{hj} is from the mean. Let C be the set of deviations $d_{h,j}$.

$$C = \{d_{hj} \mid h = \overline{1, N}\}$$

Apply the modified 2-means to C and obtain C_1 which is the expected cluster and C_2 which contains non-significant elements. Figure 6.14 summarizes the procedure using K-means clustering for the average measure values.

Now let the mean of the useful cluster C_1 be the substitute mean of C. That is, $\overline{Measure\_Data\_DM_{n+1,j}}$ is the final mean of C for each dmension member $dm_{n+1,j}$ where $j = \overline{1, n_t}$, and get $\overline{P_j}$ from (6-2) respectively.

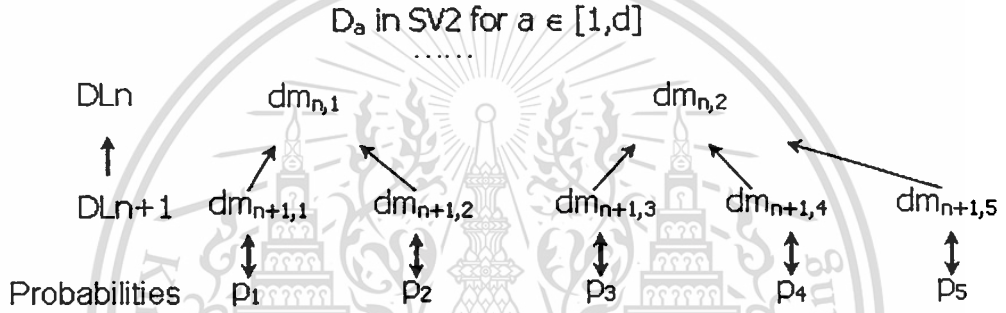


Figure 6.15 An Illustration for a General Case (SV1 and SV2 in previous parts)

A Fact in the SV1

DLn	Key1	Key2	Measure
$dm_{n,1}$	$k_{1,1}$	$k_{2,1}$	m_1
$dm_{n,1}$	$k_{1,1}$	$k_{2,2}$	m_2
$dm_{n,1}$	$k_{1,2}$	$k_{2,1}$	m_3
$dm_{n,1}$	$k_{1,2}$	$k_{2,2}$	m_4
$dm_{n,2}$	$k_{1,1}$	$k_{2,2}$	m_5
$dm_{n,2}$	$k_{1,2}$	$k_{2,2}$	m_6

A Corresponding Transformed Fact in the SV2

DLn+1	Key1	Key2	Measure
$dm_{n+1,1}$	$k_{1,1}$	$k_{2,1}$	m_1*p_1
$dm_{n+1,2}$	$k_{1,1}$	$k_{2,1}$	m_1*p_2
$dm_{n+1,1}$	$k_{1,1}$	$k_{2,2}$	m_2*p_1
$dm_{n+1,2}$	$k_{1,1}$	$k_{2,2}$	m_2*p_2
....	.....	....	.....
$dm_{n+1,3}$	$k_{1,2}$	$k_{2,2}$	m_6*p_3
$dm_{n+1,4}$	$k_{1,2}$	$k_{2,2}$	m_6*p_4
$dm_{n+1,5}$	$k_{1,2}$	$k_{2,2}$	m_6*p_5

Figure 6.16 An Illustration for the Disaggregating of Measure Data in SV1 using Probabilities

Using achieved probabilities from individual measure data or from average measure data, how to apply these probabilities on inter-version transformations is shown Figures 6.15 and 6.16. For an illustration, a case study below in Figure 6.17 has one combination of all key members of dimensions Customer and Product. In general, the number of combinations of all key members of dimensions different from the dimension evolved is greater than 1. The proposed technique is flexible to be applied in practice

with the transformations on each combination of all key members of other unchanged dimensions. An inter-version transformation is demonstrated in Figures 6-16 by splitting each measure cell in a fact of SV1 according to the number of its associated dimension members at the dimension level DL_{n+1} and their respective probabilities in order to obtain estimated measure data in a corresponding fact of SV2.

6.3.3 The Evaluation on the Inter-Version Transformation Techniques

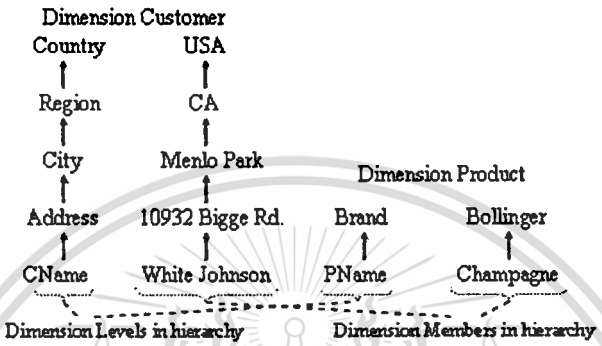


Figure 6.17 Dimensions Customer and Product unchanged in both SV1 and SV2

Table 6.1 Dimension Time in SV1

TKey	*Quarter*	*Year*
T1	Q1	1962
T2	Q2	1962
T3	Q3	1962
T4	Q4	1962
T5	Q1	1963
T6	Q2	1963
T7	Q3	1963
T8	Q4	1963

Table 6.2 Dimension Time in SV2

TKey	*Month*	*Quarter*	*Year*
T9	Jan	Q1	1964
T10	Feb	Q1	1964
T11	Mar	Q1	1964
T12	Apr	Q2	1964
...	...	...	...
T90	Oct	Q4	1969
T91	Nov	Q4	1969
T92	Dec	Q4	1969

Table 6.3 The Fact table SALE in SV1

TKey	*CName*	*PName*	*Sale*
T1	White Johnson	Champagne	8.278
T2	White Johnson	Champagne	8.703
T3	White Johnson	Champagne	7.416
T4	White Johnson	Champagne	17.197
T5	White Johnson	Champagne	8.047
T6	White Johnson	Champagne	10.272
T7	White Johnson	Champagne	8.382
T8	White Johnson	Champagne	19.669

Table 6.4 The Fact table SALE in SV2

TKey	*CName*	*PName*	*Sale*
T9	White Johnson	Champagne	3.113
...	...	...	...
T20	White Johnson	Champagne	9.254
T21	White Johnson	Champagne	5.375
T22	White Johnson	Champagne	3.088
T23	White Johnson	Champagne	3.718
T24	White Johnson	Champagne	4.514
T25	White Johnson	Champagne	4.52
T26	White Johnson	Champagne	4.539
T27	White Johnson	Champagne	3.663
T28	White Johnson	Champagne	1.643
T29	White Johnson	Champagne	4.739
T30	White Johnson	Champagne	5.428
T31	White Johnson	Champagne	8.314
T32	White Johnson	Champagne	10.651
T33	White Johnson	Champagne	3.633
....	....	...	...
T44	White Johnson	Champagne	11.331
T45	White Johnson	Champagne	4.016
...	....	....	...
T56	White Johnson	Champagne	13.916
T57	White Johnson	Champagne	4.016
...	...	....	...
T68	White Johnson	Champagne	13.916
T69	White Johnson	Champagne	2.639
...	....	....	...
T80	White Johnson	Champagne	13.076
T81	White Johnson	Champagne	3.934
...	...	...	...
T92	White Johnson	Champagne	12.67

Consider a temporal data warehouse TDW with two structure versions including SV1 valid in [01/01/1962, 31/12/1963] and SV2 in [01/01/1964, 31/12/1969]. The

Table 6.5 The Output Fact SALE in SV2 according to the Classical Techniques

TKey	*CName*	*PName*	*Weighting Factor*	*Sale*
T1.1	White Johnson	Champagne	0.333333	2.759333
T1.2	White Johnson	Champagne	0.333333	2.759333
T1.3	White Johnson	Champagne	0.333333	2.759333
T1.4	White Johnson	Champagne	0.333333	2.901
T1.5	White Johnson	Champagne	0.333333	2.901
T1.6	White Johnson	Champagne	0.333333	2.901
T1.7	White Johnson	Champagne	0.333333	2.472
....	...	....	...	....
T1.10	White Johnson	Champagne	0.333333	5.732333
...	...	...	...	...
T1.13	White Johnson	Champagne	0.333333	2.682333
....	...	...	...	...
T1.16	White Johnson	Champagne	0.333333	3.424
....	....	...	...	...
T1.19	White Johnson	Champagne	0.333333	2.794
....	...	...	....	....
T1.22	White Johnson	Champagne	0.333333	6.556333
T1.23	White Johnson	Champagne	0.333333	6.556333
T1.24	White Johnson	Champagne	0.333333	6.556333

Table 6.6 The Output Fact SALE in SV2 according to the First Approach of Ours

TKey	*CName*	*PName*	*Individual Proportion*	*Sale*
T1.1	White Johnson	Champagne	0.441261	3.652758
T1.2	White Johnson	Champagne	0.25351	2.098552
T1.3	White Johnson	Champagne	0.305229	2.526689
T1.4	White Johnson	Champagne	0.332572	2.894374
T1.5	White Johnson	Champagne	0.333014	2.898221
T1.6	White Johnson	Champagne	0.334414	2.910404
T1.7	White Johnson	Champagne	0.364659	2.704311
T1.8	White Johnson	Champagne	0.163564	1.21299
T1.9	White Johnson	Champagne	0.471777	3.498698
T1.10	White Johnson	Champagne	0.222523	3.826726
T1.11	White Johnson	Champagne	0.340835	5.861348
T1.12	White Johnson	Champagne	0.436642	7.508927
T1.13	White Johnson	Champagne	0.441261	3.550827
T1.14	White Johnson	Champagne	0.25351	2.039991
T1.15	White Johnson	Champagne	0.305229	2.456181
T1.16	White Johnson	Champagne	0.332572	3.41618
T1.17	White Johnson	Champagne	0.333014	3.420721
T1.18	White Johnson	Champagne	0.334414	3.4351
T1.19	White Johnson	Champagne	0.364659	3.056572
T1.20	White Johnson	Champagne	0.163564	1.370993
T1.21	White Johnson	Champagne	0.471777	3.954435
T1.22	White Johnson	Champagne	0.222523	4.376802
T1.23	White Johnson	Champagne	0.340835	6.703893
T1.24	White Johnson	Champagne	0.436642	8.588305

warehouse data structure of TDW at the current detail level is organized using a star schema with a fact SALE and three dimensions, namely Time, Customer, and Product for both SV1 and SV2. Let the dimensions Customer and Product be below unchanged in both SV1 and SV2 in Figure 6.17, the dimension Time be changed by adding the

dimension level Month in SV1 supporting quarterly data in Table 6.1 to gain SV2 supporting monthly data in Table 6.2 for more detailed data on their business. All sample and measure data for this case study are from [13]. For the sake of the illustration clarity and the space, there is one record given in the unchanged dimension tables Customer and Product. The measure data in the fact tables SALE in Table 6.3 for SV1 and in Table 6.4 for SV2 show the number of bottles per 1000 bought by a customer at a point in time.

According to their techniques, all weighting factors for their transformation functions are 1/3 in Table 6.5 because a quarter consists of three months. In order words, each dimension member at the dimension level DLn (Quarter) is associated with three dimension members at the dimension level DLn+1 (Month) in SV2 when the dimension level DLn+1 (Month) is added under the dimension level DLn (Quarter) into the dimension Time in SV1.

For the first approach of our proposed technique, measure data in SV1 are transformed into the structure of SV2 using probabilities calculated with measure data in the structure of SV2 in 1965 as shown in Table 6.6.

Applying the second approach to the case study, the output is shown in Table 6.7.

Table 6.7 The Output Fact SALE in SV2 according to the Second Approach of Ours

TKey	*CName*	*PName*	*Average Proportion*	*Sale*
T1.1	White Johnson	Champagne	0.341022	2.82298
T1.2	White Johnson	Champagne	0.286382	2.370668
T1.3	White Johnson	Champagne	0.372596	3.084353
T1.4	White Johnson	Champagne	0.321033	2.793947
T1.5	White Johnson	Champagne	0.334304	2.909446
T1.6	White Johnson	Champagne	0.344664	2.999607
T1.7	White Johnson	Champagne	0.362787	2.690425
T1.8	White Johnson	Champagne	0.159652	1.183976
T1.9	White Johnson	Champagne	0.477562	3.541599
T1.10	White Johnson	Champagne	0.233693	4.018813
T1.11	White Johnson	Champagne	0.339002	5.829823
T1.12	White Johnson	Champagne	0.427305	7.348364
T1.13	White Johnson	Champagne	0.341022	2.744204
T1.14	White Johnson	Champagne	0.286382	2.304514
T1.15	White Johnson	Champagne	0.372596	2.998283
T1.16	White Johnson	Champagne	0.321033	3.297647
T1.17	White Johnson	Champagne	0.334304	3.433968
T1.18	White Johnson	Champagne	0.344664	3.540384
T1.19	White Johnson	Champagne	0.362787	3.040877
T1.20	White Johnson	Champagne	0.159652	1.338199
T1.21	White Johnson	Champagne	0.477562	4.002924
T1.22	White Johnson	Champagne	0.233693	4.596502
T1.23	White Johnson	Champagne	0.339002	6.667837
T1.24	White Johnson	Champagne	0.427305	8.404662

For the comparison between two approaches of our technique to the existing approach of the classical techniques, we learn how dissimilar the estimated data in SV2 is to the actual monthly data in SV1 using the Euclidean distance. Given datasets $X = \{x_i \mid i = \overline{1,N}\}$ for the actual data, $Y = \{y_i \mid i = \overline{1,N}\}$ for the estimated data, the dissimilarity of the estimated data to the actual data is computed.

$$\text{dissimilarity } (X,Y) = \sqrt{\sum_{i=1}^{N}(y_i - x_i)^2} \qquad (6\text{-}6)$$

The smaller Euclidean distance, the less dissimilar the estimated data is to actual data; i.e. the smaller is the better.

Let X be the actual data, Y be the estimated data from classical techniques, Z be the estimated data with probabilities calculated using the individual monthly data in 1965, V be the estimated data with probabilities calculated using the average measure data from the K-means clustering technique. All estimated measure data are put together in Table 6.8. The dissimilarities from estimated data to actual data are computed by using (6-6) and a comparison is shown in Figures 6.18 and 6.19.

dissimilarity (X, Y) = 3.765654

dissimilarity (X, Z) = 2.247014

dissimilarity (X, V) = 1.619257

Table 6.8 Comparison of Estimated Results to Actual Data

Quarterly Data	*Actual Monthly Data*	*Classical Estimated Data*	*Average Estimated Data*	*Individual Estimated Data*
	2.851	2.759333	2.82298	3.652758
8.278	2.672	2.759333	2.370668	2.098552
	2.755	2.759333	3.084353	2.526689
	2.721	2.901	2.793947	2.894374
8.703	2.946	2.901	2.909446	2.898221
	3.036	2.901	2.999607	2.910404
	2.282	2.472	2.690425	2.704311
7.416	2.212	2.472	1.183976	1.21299
	2.922	2.472	3.541599	3.498698
	4.301	5.732333	4.018813	3.826726
17.197	5.764	5.732333	5.829823	5.861348
	7.132	5.732333	7.348364	7.508927
	2.541	2.682333	2.744204	3.550827
8.047	2.475	2.682333	2.304514	2.039991
	3.031	2.682333	2.998283	2.456181
	3.266	3.424	3.297647	3.41618
10.272	3.776	3.424	3.433968	3.420721
	3.23	3.424	3.540384	3.4351
	3.028	2.794	3.040877	3.056572
8.382	1.759	2.794	1.338199	1.370993
	3.595	2.794	4.002924	3.954435
	4.474	6.556333	4.596502	4.376802
19.669	6.838	6.556333	6.667837	6.703893
	8.357	6.556333	8.404662	8.588305

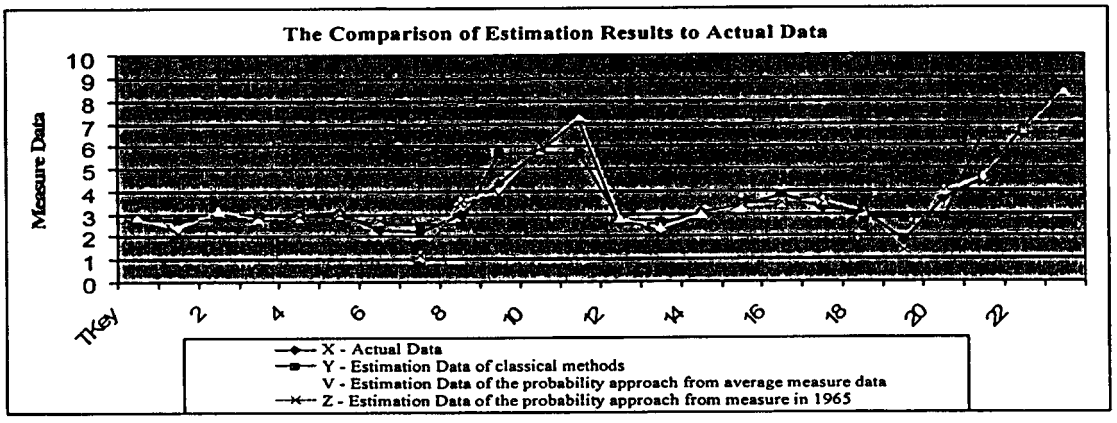


Figure 6.18 A Graph for the Comparison of Estimated Results to Actual Data in Table 6.8

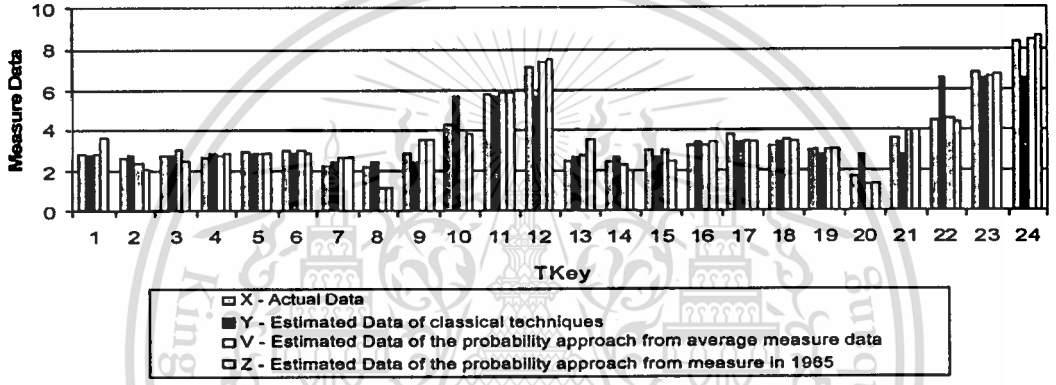


Figure 6.19 A corresponding Chart for the Comparison of Estimated Results to Actual Data in Table 6.8

For the following two examples from [38], we just concentrate on pre- and post-transformed measure data together with the affected levels and their respective probabilities.

The first monthly sample data from table B.7 Monthly Sales of a 32-oz Soft Drink in [Jan 1972, Dec 1975] are shown in Table 6.9. Individual measure data in the year 1975 are used to calculate individual probabilities in our first approach. All weighting factors and probabilities are achieved in Table 6.10.

The dissimilarities from estimated data to actual data are below computed.

Dissimilarity (X,Y) for classical methods = 14

Dissimilarity (X,Z) for probabilities from individual measure data = 2.417808

Dissimilarity (X,V) for probabilities from average measure data = 4.003233

Table 6.9 The First Monthly Sample Data in SV2

	1973	1974	1975
Jan	35	45	52
Feb	40	49	60
Mar	46	57	66
Apr	55	68	80
May	60	78	85
Jun	68	80	95
Jul	72	88	100
Aug	75	90	104
Sep	70	84	101
Oct	66	80	94
Nov	58	57	81
Dec	50	60	70

Table 6.10 Weighting Factors, and Probabilities

Actual Data	Classical	Average	Individual
28	0.333333	0.293333	0.292135
31	0.333333	0.331111	0.337079
36	0.333333	0.375556	0.370787
43	0.333333	0.303438	0.307692
46	0.333333	0.333333	0.326923
52	0.333333	0.363229	0.365385
55	0.333333	0.331633	0.327869
59	0.333333	0.343112	0.340984
58	0.333333	0.325255	0.331148
55	0.333333	0.38961	0.383673
47	0.333333	0.318182	0.330612
40	0.333333	0.292208	0.285714

The transformed measure data are achieved in Table 6.11 and in Figure 6.20.

Table 6.11 The First Measure Data of SV1 Transformed into SV2

Quarterly Data	Actual Monthly Data	Classical Techniques	Average Choice	Individual Choice
	28	31.666667	27.866667	27.752809
	31	31.666667	31.455556	32.022472
95	36	31.666667	35.677778	35.224719
	43	47	42.784753	43.384615
	46	47	47	46.096154
141	52	47	51.215247	51.519231
	55	57.333333	57.040816	56.393443
	59	57.333333	59.015306	58.64918
172	58	57.333333	55.943878	56.957377
	55	47.333333	55.324675	54.481633
	47	47.333333	45.181818	46.946939
142	40	47.333333	41.493506	40.571429

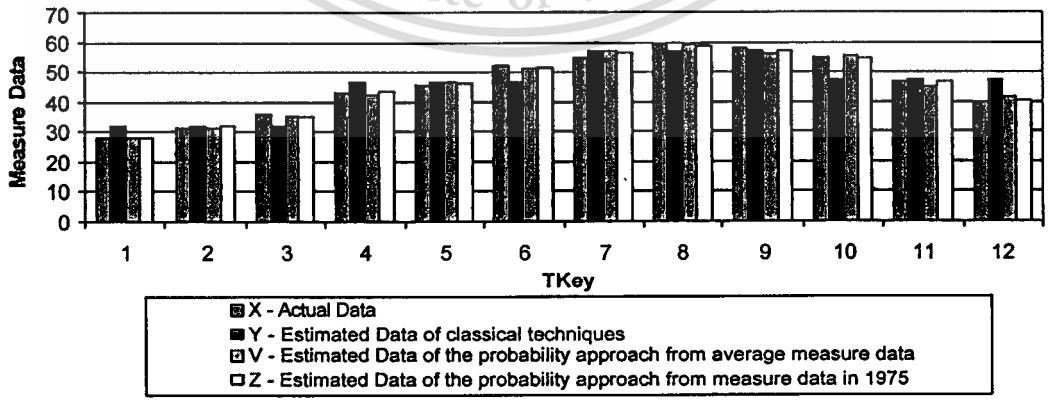


Figure 6.20 A corresponding Chart for the Comparison of Estimated Results to Actual Data in Table 6.11

Table 6.9 The First Monthly Sample Data in SV2

	1973	1974	1975
Jan	35	45	52
Feb	40	49	60
Mar	46	57	66
Apr	55	68	80
May	60	78	85
Jun	68	80	95
Jul	72	88	100
Aug	75	90	104
Sep	70	84	101
Oct	66	80	94
Nov	58	57	81
Dec	50	60	70

Table 6.10 Weighting Factors, and Probabilities

Actual Data	Classical	Average	Individual
28	0.333333	0.293333	0.292135
31	0.333333	0.331111	0.337079
36	0.333333	0.375556	0.370787
43	0.333333	0.303438	0.307692
46	0.333333	0.333333	0.326923
52	0.333333	0.363229	0.365385
55	0.333333	0.331633	0.327869
59	0.333333	0.343112	0.340984
58	0.333333	0.325255	0.331148
55	0.333333	0.38961	0.383673
47	0.333333	0.318182	0.330612
40	0.333333	0.292208	0.285714

The transformed measure data are achieved in Table 6.11 and in Figure 6.20.

Table 6.11 The First Measure Data of SV1 Transformed into SV2

Quarterly Data	Actual Monthly Data	Classical Techniques	Average Choice	Individual Choice
	28	31.666667	27.866667	27.752809
	31	31.666667	31.455556	32.022472
95	36	31.666667	35.677778	35.224719
	43	47	42.784753	43.384615
	46	47	47	46.096154
141	52	47	51.215247	51.519231
	55	57.333333	57.040816	56.393443
	59	57.333333	59.015306	58.64918
172	58	57.333333	55.943878	56.957377
	55	47.333333	55.324675	54.481633
	47	47.333333	45.181818	46.946939
142	40	47.333333	41.493506	40.571429

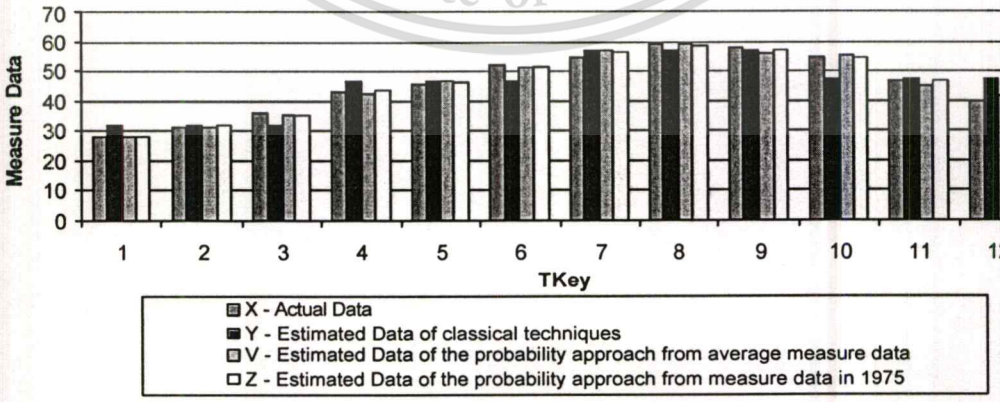


Figure 6.20 A corresponding Chart for the Comparison of Estimated Results to Actual Data in Table 6.11

The second monthly sample data from Series E. for Industrial Production in Spain for 5 years [Makridakis 868] are shown in Table 6.12. Individual measure data in the fifth year are used to calculate individual probabilities in our first approach. All weighting factors and probabilities are achieved in Table 6.13.

Table 6.12 The Second Monthly Data in SV2

	Year 2	Year 3	Year 4	Year 5
Jan	141	132	148	161
Feb	156	139	137	160
Mar	151	139	137	167
Apr	160	137	155	178
May	156	144	152	167
Jun	160	146	153	176
Jul	161	149	152	173
Aug	149	142	153	164
Sep	118	101	113	123
Oct	147	141	151	175
Nov	158	122	159	175
Dec	146	145	165	176

Table 6.13 Weighting Factors, and Probabilities

Actual Data	Classical	Average	Individual
128	0.333333	0.327664	0.329918
134	0.333333	0.335601	0.327869
133	0.333333	0.336735	0.342213
141	0.333333	0.336538	0.341651
134	0.333333	0.32906	0.320537
142	0.333333	0.334402	0.337812
143	0.333333	0.37331	0.376087
136	0.333333	0.355085	0.356522
108	0.333333	0.271605	0.267391
142	0.333333	0.321814	0.3327
146	0.333333	0.342333	0.3327
149	0.333333	0.335853	0.334601

The transformed data of SV1 into SV2 are presented in Table 6.14 and Figure 6.21. The respective dissimilarities of estimated data to actual data are below calculated.

Dissimilarity (X,Y) for classical methods = 27.736859

Dissimilarity (X,Z) for probabilities from individual measure data = 9.176997

Dissimilarity (X,V) for probabilities from average measure data = 7.331597

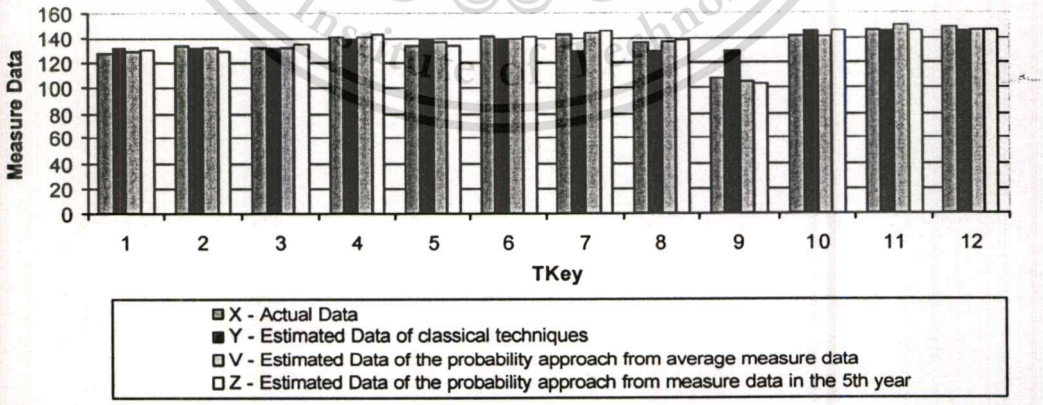


Figure 6.21 A corresponding Chart for the Comparison of Estimated Results in Table 6.14

Table 6.14 The Second Measure Data of SV1 Transformed into SV2

Quarterly Data	Actual Monthly Data	Classical Techniques	Average Choice	Individual Choice
	128	131.666667	129.427438	130.317623
	134	131.666667	132.562358	129.508197
395	133	131.666667	133.010204	135.17418
	141	139	140.336538	142.46833
	134	139	137.217949	133.664107
417	142	139	139.445513	140.867562
	143	129	144.470899	145.545652
	136	129	137.417989	137.973913
387	108	129	105.111111	103.480435
	142	145.666667	140.632829	145.389734
	146	145.666667	149.599352	145.389734
437	149	145.666667	146.767819	146.220532

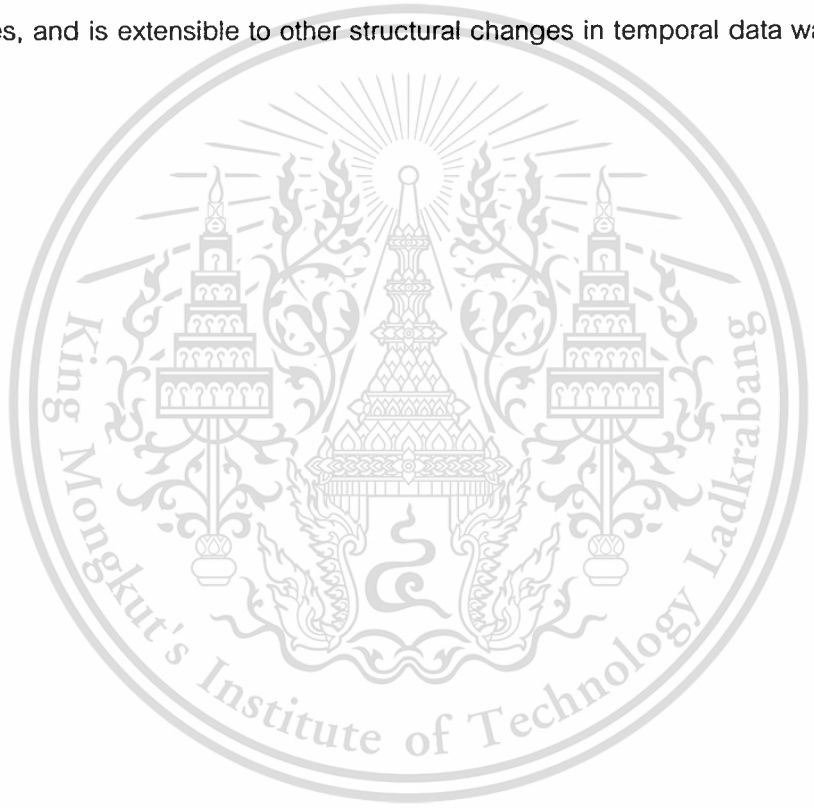
From comparisons above, the dissimilarities show that our proposed technique provides an initiative schema transformation with more precise results although the results calculated by using probabilities are rather dependent on the sample data in the target structure version into which the measure data is going to be transformed. Furthermore, in spite of the uncertain property of sample data from the users to calculate proportion, we believe that good sample data is easily found from the huge volume of data in the warehouse over time by experienced analysts, statisticians, and warehouse administrators according to the nature of their organizational data. Therefore, the proposed schema transformation technique is useful and applicable.

6.4 Summary

An overall review on warehouse schema evolutions has been briefly given so that warehouse schema evolutions of adding/removing a dimension and adding/removing a dimension level are chosen to be tackled. For this choice, transformations among versions are required. Moreover, how to support users to describe their warehouse schema evolutions has been explained through a pre-defined deep-structured natural language. This approach will take advantage of the approach to the design of a warehouse schema based on the NIAM/ORM modeling in the previous chapter.

Apart from the management of schema versioning, this chapter has presented a novel technique to deal with schema evolutions of a temporal data warehouse. All the changes at the schema level are converted into data changes at the instance level.

Proposed transformation functions using probabilities are applied on measure data of one warehouse structure version that is presented in the different structure of another warehouse structure version after schema transformations. By carrying out such transformations, the whole data of a temporal data warehouse are consistent in one selected structure version among many structure versions. Furthermore, it is possible to make queries and analyze data on a long time interval that includes several valid intervals of some warehouse structure versions. The result from the proposed technique was found to be more precise than those from other techniques. Above all, our approach does not depend on the existence of the dimension Time in temporal data warehouses, and is extensible to other structural changes in temporal data warehouses as well.



Chapter 7

The Implementation of the Architecture TIWA1

The temporal information warehouse which has designed and developed with the proposed architecture TIWA1 was implemented using the Microsoft Visual .Net technology for programming and Microsoft SQL Server 2000 for our database server as well as for output user temporal warehouses. This chapter will discuss some issues included into the implementation of our system.

7.1 System Requirements

Both server and client sides are required for the Microsoft .Net framework.

Server Side:

- Operating System: Windows Server 2000
- Database Server: Microsoft SQL Server 2000 Enterprise Edition
- Extra Component: Microsoft SQL Server 2000 Analysis Services / Service Pack 3

Client Side (Output User Temporal Warehouses from our System):

- Database: Microsoft SQL Server 2000 (Standard Edition, Personal Edition)

7.2 The Lexical Analyzers and Parsers

Due to the use of the Microsoft .Net technology, the lexical analyzers and parsers built using the Flex/Bison tools are unable to directly interact with main programs written in any .Net languages. In order to cope with this issue, the Shell function that allows us to launch executable programs within Visual Studio was used. The following steps are performed to obtain results from the Lexical Analyzers and Parsers for the design of a warehouse schema and the design of warehouse schema versioning through our deep-structured natural language interface.

- Generate the lexical analyzers by using Flex and the parsers by Bison.
- Combine these two output files *.c into a Visual C++ project to gain an executable program *.exe that receives a text file as its input and produces other text files as its output. Those output text files are understandable in terms of NIAM/ORM elements.

In other words, those files are organized to reconstruct a NIAM/ORM conceptual schema that is exactly the same as the one designed using Microsoft Visio.

- Activate the executable program *.exe by invoking the Shell function
- Once an expected output is a text file, our program developed within the .Net framework is able to read the output and do further processing like saving the information of a conceptual schema into 5NF meta tables, transforming a conceptual schema into a relational schema, and transforming a conceptual schema into a warehouse schema.

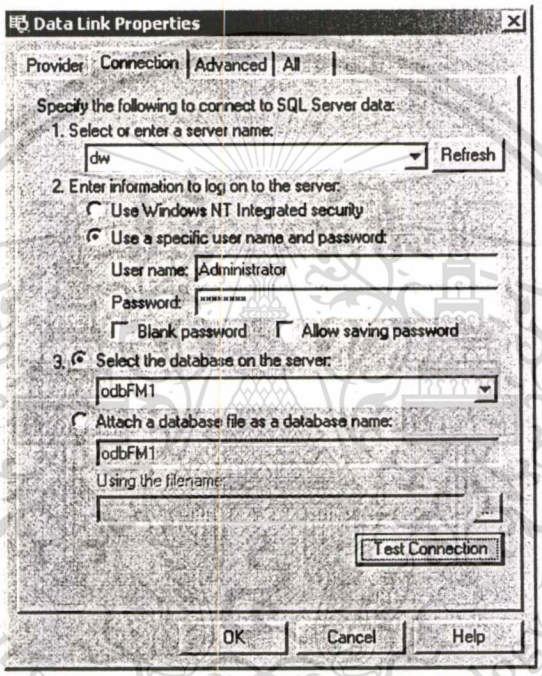


Figure 7.1 The Data Link Properties Dialog Box for the Connections to Data Sources

7.3 Interactions with Data Sources

- For connections to operational data sources from which data are put into a warehouse; and for connections to data sources that keep data samples for the calculation of missing data filling parameters and of measure data transforming parameters, the Data Link Properties dialog box in Figure 7.1 that is the standard Windows system interface for configuring connection strings to data sources is employed. We activate this dialog box that is a universal data link (.udl) file by using an executable file *.bat. However, in the implementation, only the connections to data sources Microsoft SQL Server are supported.

- Summarize measure data up to the higher level DLn
- Carry out the transforming techniques on measure data detailed at the level DLn to gain measure data back detailed at the lower level DLn+1
- Compute the Euclidean distances of actual measure data at the level DLn+1 and transformed measure data at the level DLn+1 as presented in the previous test procedure 7.4.1
- Compare these achieved distances to reach evaluations on techniques in the test

7.5 The COM Component Employment

For analyzing temporal warehouse data organized according to our temporal information warehouse design, COM components are employed in the implementation of our system.

The purpose of the employment of these components is to prove that results from our system are capable of working with some available frond-end tools for online analytical processing, data mining and reporting. Three COM components are OLAP Manager Cube Browser (msmdcb.ocx), Microsoft Data Mining Model Browser (msdmbr.ocx), and Microsoft Office PivotTable 10.0 (OWC10.dll) respectively. To be able to set up input parameters for these components, Microsoft Decision Support Objects (msmddo80.dll/msmddo.dll) that is a library of COM classes and interfaces that provide access to the Analysis server managed by Microsoft SQL Server 2000 Analysis Services are included in our implementation as well.

7.6 Summary

This section has presented several dominant issues in the implementation of our system. All functions that support the design of a temporal information warehouse with schema versioning were implemented using the .Net framework and the Microsfoft SQL Server 2000 DBMS. These functions are the filling of missing measure data before the loading process, the gathering of metadata of an operational database by applying the ONF algorithm, the generating of a warehouse schema from a conceptual schema, the transforming of measure data among warehouse versions and the querying and analyzing of warehouse data in the structure of a selected version. How users play with these functions is illustrated in Appendices.

Chapter 8

Conclusion and Further Research

8.1 Conclusion

From the architecture TIWA1 based on the NIAM/ORM modeling, a temporal information warehouse with schema versioning has been designed and developed. Our system implementing this architecture supports users to provide descriptions of operational conceptual schemas in the NIAM/ORM modeling via a deep-structured natural language interface. Such a feature allows the system to make use of the design of NIAM/ORM-based operational sources from which data will be loaded into temporal information warehouses.

In addition, using this architecture, the data availability in temporal warehouses is strongly enhanced at the cleaning task with filling in missing data and at the transformation process among many warehouse structure versions. Therefore, an achieved temporal warehouse is able to be efficiently used with any existing font-end tools such as online analytical processing (OLAP) tools, data mining tools, and so on. With all consistent data prepared in the temporal warehouse, it is useful for these tools to mine warehouse data in a specific warehouse structure version with temporal queries probably spanning a long time period greater than the life span of the version in which users are currently interested.

8.2 Further Research

Even though the two proposed techniques for handling missing data and for inter-version transformations are rather simple, they obviously provided better results than other corresponding techniques. However, to obtain more and more precise estimated data for the missing problem and transformed data for the inter-version transformations is still promising by employing artificial intelligent techniques such as artificial neural networks and fuzzy clustering techniques.

Since data warehousing is a popular choice of the integration of many heterogeneous operational data sources stored under various data models for the

knowledge discovery process in decision support systems, the schema integration is very important to prepare a global model for a warehouse so that data from these sources are able to be kept together consistently in the warehouse. It is obvious to realize that this part is not initiative in data warehousing. However, the schema integration for a temporal information warehouse is quite attractive in terms of temporal semantics because it has been supported little so far. Hence, designing a global warehouse schema capable of capturing time-dependent objects in operational sources will be taken into consideration as one of future works.

Furthermore, the time-stamping of structural elements in our approach might be extended to the instance level in order to gain a so-called bi-temporal information warehouse. This aspect is dedicated to the design of a temporal information warehouse using an underlying bi-temporal database instead of the existing non-temporal relational database in this research. It was due to the concept of a temporal information warehouse at this stage "A temporal information warehouse is defined as an information warehouse with warehouse schema versioning". In the future, a bi-temporal information warehouse is going to be focused on with respect to how legacy sources support temporal features because this concern has been paid a little attention, only in the context of data warehousing, not temporal data warehousing. The bi-temporal enhancement in information warehouses will provide a lot of help on discovering interesting information for decision and prediction making based on the state of data objects within a life span of a specific warehouse version.

Finally, how a query language such as SQL-92, TSQL2 and so forth can work on a tentative bi-temporal information warehouse is necessary to be figured out. The purpose behind this issue is the warehouse will be able to be further used with some available analysis tools and reporting facilities. It will also evaluate how well the outcome from the future research works in practice.

References

[1]. A. Abello, J. Samos, F. Saltor, "A Framework for the Classification and Description of Multidimensional Data Models", In Proc. of the 12th International Conference on Database and Expert Systems Applications, Germany, 2001

[2]. A. Abello, C. Martin, A Bitemporal Storage Structure for a Corporate Data Warehouse, In Proc. of the ICEIS, France, 2003

[3]. A. Mendelzon, A. Vaisman, Temporal Queries in OLAP, In Proc. of the 26th VLDB Conference, Egypt, 2000

[4]. A. Olinsky, S. Chen, L. Harlow, The Comparative Efficacy of Imputation Methods for Missing Data in Structural Equation Modeling, European Journal of Operational Research, v. 151, Nov 16th, 2003

[5]. A. Witkowski, S. Bellamkonda, T. Bozkaya, N. Folkert, A. Gupta, L. Sheng, S. Subramanian, Business Modeling Using SQL Spreadsheets, In Proc. of the 29th VLDB Conference, Germany, 2003

[6]. B. A. Devlin, P. T. Murphy, An Architecture for a Business and Information System, IBM Systems Journal, 27(1), 1988

[7]. B. Bebel, J. Eder, C. Koncilia, T. Morzy, R. Wrembel, Creation and Management of Versions in Multiversion Data Warehouse, In Proc. of the ACM Symposium on Applied Computing, SAC, Cyprus, 2004

[8]. C. Knocilia, A Bi-Temporal Data Warehouse Model, In Proc. of the CAiSE, 2003

[9]. C. Letz, E. T. Henn, G. Vossen, Consistency in Data Warehouse Dimensions, In Proc. of the International Database Engineering and Applications Symposium, Canada, 2002

[10]. C. Martin, A. Abello, A Temporal Study of Data Sources to Load a Corporate Data Warehouse, In Proc. of the Data Warehousing and Knowledge Discovery Conference, DaWaK, Czech Republic, 2003

[11]. C. Sapia, M. Blaschka, G. Hofling, B. Dinter, "Extending the E/R Model for the Multidimensional Paradigm", In Proc. of the Workshops on Data Warehousing and Data Mining: Advances in Database Technologies, 1998

[12]. C. S. Jensen, C. E. Dyreson, editors, A Consensus Glossary of Temporal Database Concepts – Feb. 1998 Version (Springer-Verlag, 1998)

[13]. D. C. Montgomery, L. A. Johnson, J. S. Gardiner, Forecasting & Time Series Analysis (McGraw-Hill, Inc. 1990)

[14]. E. A. Rundensteiner, A. Koeller, X. Zhang, Maintaining Data Warehouses over Changing Information Sources, Communications of the ACM, 2000

[15]. F. Ravat, O. Teste, G. Zurflush, Towards Data Warehouse Design, In Proc. of the 8th International Conference on Information and Knowledge Management, USA, 1999

[16]. G. E. A. P. A. Batista, M. C. Monard, An Analysis of Four Missing Data Treatment Methods for Supervised Learning, In Proc. of The 1st International Workshop on Data Cleaning and Preprocessing, Maebashi, 2002

[17]. G. E. A. P. A. Batista, M. C. Monard, A Study of K-Nearest Neighbor as an Imputation Method, In Proc. of the 2nd International Conference on Hybrid Intelligent Systems, Chile, 2002

[18]. G. M. Nijssen, T. A. Halpin, Conceptual Schema and Relational Database Design, (Prentice Hall, 1989)

[19]. J. Eder, C. Koncilia, T. MOrzy, A Model for a Temporal Data Warehouse, In Proc. of the International Workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations, Italy, 2001

[20]. J. Eder, C. Koncilia, Representing Knowledge about Changes in Data Warehouse Structures, In Proc. of the 3rd Workshop on Knowledge Management in Electronic Government, Denmark, 2002

[21]. J. Eder, C. Koncilia, H. Kogler, Temporal Data Warehousing: Business Cases and Solutions, In Proc. of the ICEIS, 2002

[22]. J. Eder, C. Koncilia, T. Morzy, The COMET Metamodel for Temporal Data Warehouses, In Proc. of the 14th International Conference on Advanced Information Systems Engineering, Canada, 2002

[23]. J. Han, M. Kamber, Data Mining: Concepts and Techniques (Morgan Kaufmann, 2001)

[24]. J. Trujillo, M. Palomar, "An Object-Oriented Approach to Multidimensional Database Conceptual Modeling", In Proc. of the ACM DOLAP 98 Workshop, 1998

[25]. J. Trujillo, M. Palomar, J. Gomez, I. Song, "Designing Data Warehouses with OO Conceptual Models", IEEE Computer, special issues on Data Warehouses, 34(12):66-75, 2001

[26]. J. W. Grzymala-Busse, M. Hu, A Comparison of Several Approaches to Missing Attributes Values in Data Mining, In Proc. of the 2nd International Conference on Rough Sets and Current Trends in Computing, Canada, p. 340-347, 2000

[27]. J. Widom, Research Problems in Data Warehousing, In Proc. of the 1995 International Conference on Information and Knowledge Management, 1995

[28]. K. Sattler, E. Schallehn, A Data Preparation Framework Based on a Multidatabase Language, In Proc. of the International Database Engineering and Applications Symposium, France, 2001

[29]. L. Bækgaard, "Event-Entity-Relationship Modeling in Data Warehouse Environments", In Proc. of the ACM DOLAP 99 Workshop, Missouri, 1999

[30]. M. Blaschka, C. Sapia, G. Hofling, On Schema Evolution in Multidimensional Databases, In Proc. of the 1st International Conference on Data Warehousing and Knowledge Discovery, 1999

[31]. M. Body, M. Miquel, Y. Bedard, A. Tchounikine, A Multidimensional and Multiversion Structure for OLAP Applications, In Proc. of the 5th ACM International Workshop on Data Warehousing and OLAP, USA, 2002

[32]. M. Body, M. Miquel, Y. Bedard, A. Tchnounikine, Handling Evolutions in Multidimensional Structures, In Proc. of the 19th International Conference on Data Engineering, India, 2003

[33]. M. Golfarelli, D. Maio, S. Rizzi, "Conceptual Design of Data Warehouses from E/R Schemes", In Proc. of the 31st Hawaii Intl. Conf. on System Sciences, Hawaii, 1998

[34]. M. H. Dunham, Data Mining: Introductory and Advanced Topics (Pearson Education, Inc. 2003)

[35]. N. L. Sarda, Temporal Issues in Data Warehouse Systems, In Proc. of the 1999 International Symposium on Database Applications in Non-Traditional Environments, Japan, 1999

[36]. N. Tryfona, F. Busborg, J. G. B. Christiansen, "starER: A Conceptual Model for Data Warehouse Design", In Proc. of the ACM DOLAP 99 Workshop, Missouri, 1999

[37]. P. Chountas, C. Vasilakis, E. El-Darzi, I. Petrounias, A. Tseng, Data Warehouses – TOLAP – Decision Making, In Proc. the IEEE International Conference on Systems, Man and Cybernetics, 2003

[38]. P. J. Brockwell, R. A. Davis, Time Series: Theory and Methods – Springer Series in Statistics – Second Edition (Springer-Verlag, Inc. 1991)

[39]. P. Jermyn, M. Dixon and B. J Read, Preparing Clean Views of Data for Data Mining, In Proc. of the 12th ERCIM Workshop on Database Research, Amsterdam, 2-3 Nov 1999

[40]. P. O'Neil, G. Graefe, "Multi-table joins through bitmapped join indices", ACM SIGMOD Record, Sept 1995

[41]. P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, ARKTOS: A Tool for Data Cleaning and Transformation in Data Warehouse Environments, IEEE Data Engineering Bulletin, vol. 23, no. 4, Dec 2000

[42]. R. Bliujute, S. Saltenis, G. Slivinskas, C. S. Jensen, Systematic Change Management in Dimensional Data Warehousing, In Proc. of the 3rd International Baltic Workshop on DB and IS, 1998

[43]. R. Kimball, The Data Warehouse Toolkit (John Wiley & Sons Inc., 1996)

[44]. R. M. Bruckner, B. List, J. Schiefer, A. M. Tjoa, Modeling Temporal Consistency in Data Warehouses, In Proc. of the 12th International Workshop on Database and Expert Systems Applications, 2001

[45]. S. Chaudhuri, U. Dayal, An Overview of Data Warehousing and OLAP Technology, ACM SIGMOD Record, 1997

[46]. S. D. Amo, M. H. F. Alves, Efficient Maintenance of Temporal Data Warehouses, In Proc. of the International Database Engineering and Applications Symposium, Japan, 2000

[47]. S. Englert, "Nonstop SQL scalability and availability for decision support", ACM SIGMOD Record, June 1994

[48]. S. Luja'n-Mora, J. Trujillo, P. Vassiliadis, "Advantages of UML for Multidimensional Modeling", In Proc. of the 6th International Conference on Enterprise Information Systems, Portugal, 2004

[49]. S. Luja'n-Mora, J. Trujillo, Il-Yeol Song, "Extending the UML for Multidimensional Modeling", In Proc. of the 5th International Conference on the Unified Modeling Language – the Language and its Applications, Germany, 2002

[50]. S. Luja'n-Mora, J. Trujillo, I. Song, "Multidimensional Modeling with UML Package Diagrams", In Proc. of the 21st International Conference on Conceptual Modeling, Finland, 2002

[51]. S. Luja'n-Mora, P. Vassiliadis, J. Trujillo, "Data Mapping Diagrams for Data Warehouse Design with UML", In Proc. of the 23rd International Conference on Conceptual Modeling, China, 2004

[52]. T. A. Halpin, Information Modeling and Relational Databases (Morgan Kaufmann Publishers, 2001)

[53]. T. Morzy, R. Wrembel, On Querying Versions of Multiversion Data Warehouse, In Proc. of the 7th ACM International Workshop on Data Warehousing and OLAP, DOLAP, USA, 2004

[54]. T. Tanawong, S. Chittayasothorn, "A Fact-Based Object-Oriented Database Design Tool with a Natural Language Interface", In Proc. of the TENCON, 2000

[55]. W. H. Inmon, Building the Data Warehouse (John Wiley & Sons, Inc. 1996)

[56]. W. H. Inmon, Managing the Data Warehouse (John Wiley & Sons, Inc. 1997)

[57]. Vo Thi Ngoc Chau, Suphamit Chittayasothorn, "A Missing Data Handling Technique for Data Warehouses", In Proc. of the AISTA 2004 International Conference on Advances in Intelligent Systems: Theory and Applications, in cooperation with the IEEE Computer Society, Luxembourg, Nov 15-18, 2004

[58]. Vo Thi Ngoc Chau, Suphamit Chittayasothorn, "A Schema Transformation Technique for Temporal Data Warehouses", In Proc. of the IASTED International Conference on Databases and Applications DBA 2005, Austria, Feb 14-16, 2005

[59]. Vo Thi Ngoc Chau, Suphamit Chittayasothorn, "The Design and Implemenation of an ORM-based Information Warehouse", In Proc. of the IASTED International Conference on Databases and Applications DBA 2005, Austria, Feb 14-16, 2005 (accepted)

Appendix A

User Interfaces

This section aims at an introduction to user interfaces for the design and development of a temporal information warehouse. These user interface descriptions will help users play with our system more efficiently and comfortably.

A.1 Main Interfaces

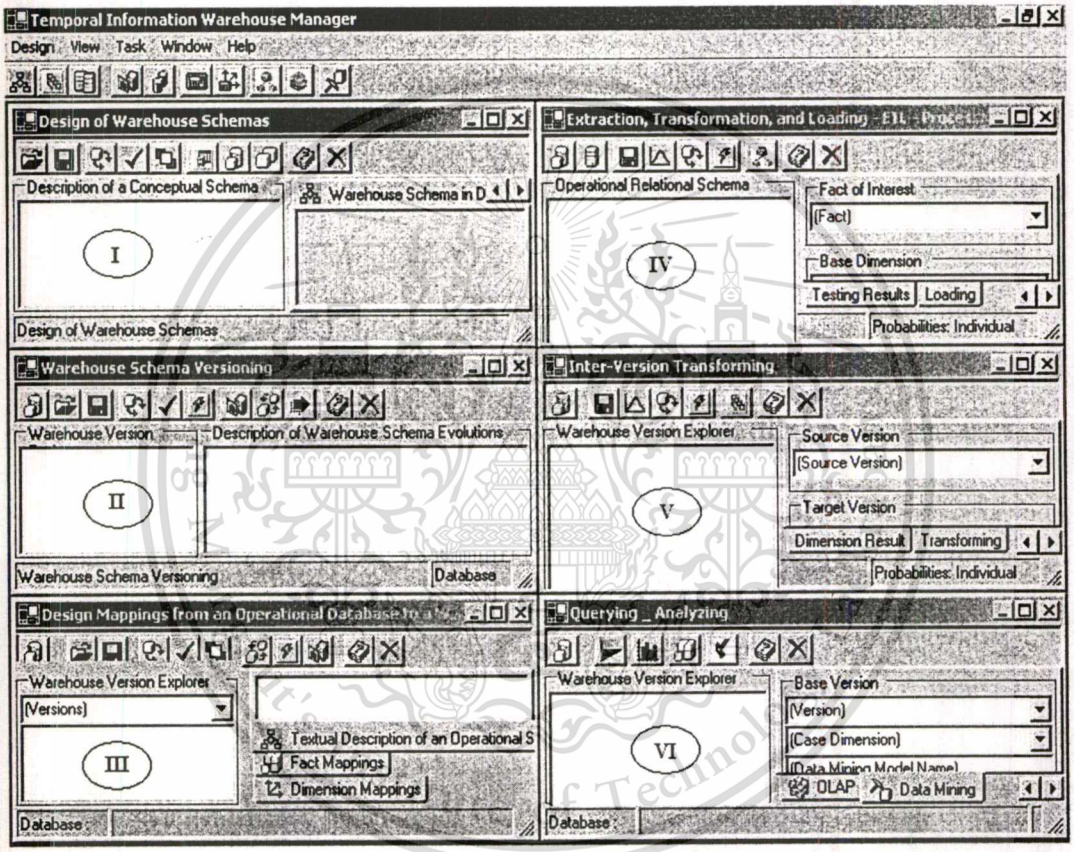


Figure A.1 Main Interfaces

Main interfaces available for users to interact with our system compose of six main dialogs corresponding to six implemented functions.

(I) The dialog box Design of Warehouse Schemas is invoked by clicking the button  .

(II) The dialog box Warehouse Schema Versioning is invoked by clicking the button  .

(III) The dialog box Design Mappings from an Operation Database to a Warehouse Version is invoked by clicking the button  .

(IV) The dialog box Extraction, Transformation, and Loading – ETL – Processing is invoked by clicking the button  or the button  .

(V) The dialog box Inter-Version Transforming is invoked by clicking the button  or the button  .

(VI) The dialog box Querying_Analyzing is invoked by clicking the button  .

These dialogs are able to be invoked by clicking appropriate menus from the main menu. Functions and characteristics of each dialog box are below described in detail.

A.2 The dialog box Design of Warehouse Schemas

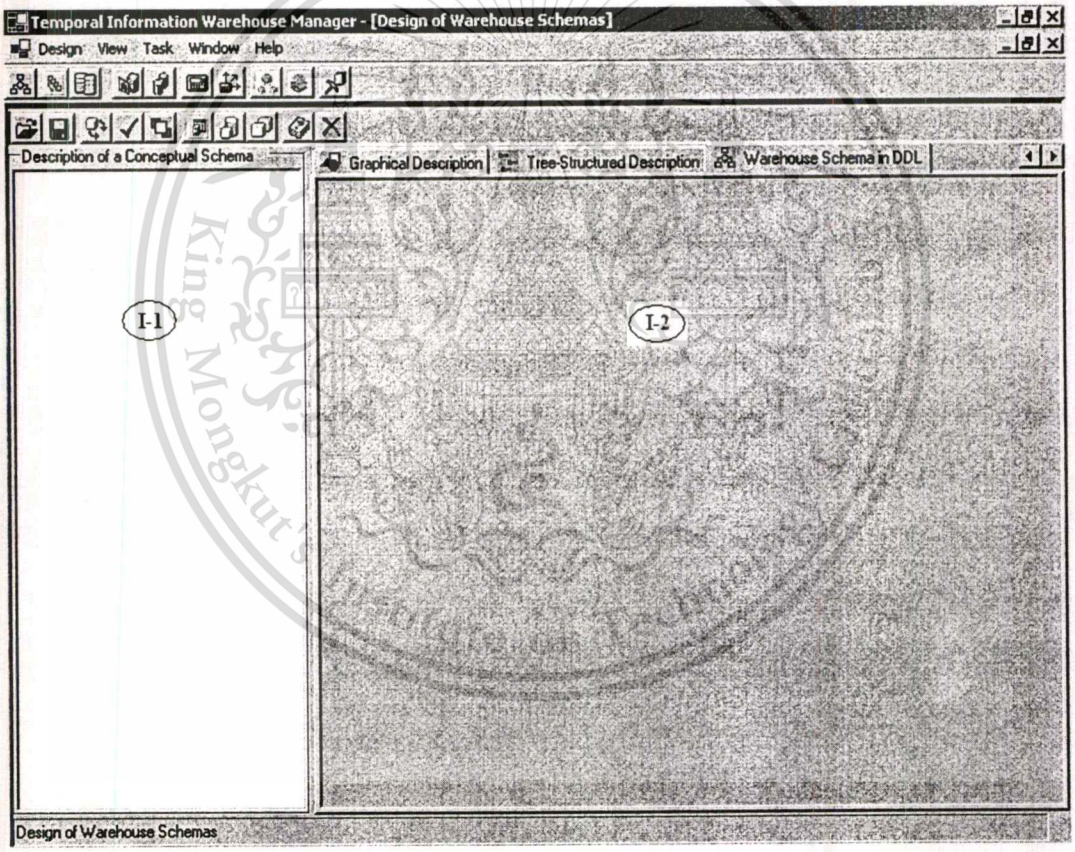


Figure A.2 The Dialog Box Design of Warehouse Schemas

(I-1) Users provide our system with a textual decription of a NIAM/ORM conceptual schema. The description might be typed interactively in the text box (I-1) or might be automatically loaded by clicking the button  to open the dialog box OpenFileDialog for an available text file containing the description.

(I-2) The output warehouse schema for the generation of table structures including fact structures and dimension structures is given in the tabpage Warehouse Schema in DDL. Besides the view of a warehouse schema in DDL, fact structures and dimension structures are organized in the tree view in the tabpage Tree-Structured Description where hierarchies among dimension levels in each dimension are clearly described.

The following are actions on this dialog box to design a warehouse schema from a conceptual schema of an operational database.

- Save the input in (I-1) into a text file: click 
- Invoke the lexical analyzer and the parser: click 
- Check the result from the lexical analyzing and parsing process: click 
- Process warehouse schema transformations if the previous action is successfully accomplished: click 
- Save the information of the input conceptual schema and the information of the output warehouse schema into 5NF meta tables: click 
- Generate warehouse table structures consisting of fact structures and dimension structures with their primary keys handled by users: click 
- Generate warehouse table structures consisting of fact structures and dimension structures with primary keys automatically handled by our system for inter-version transformations: click  . These primary keys are obtained from user primary keys and added the information about the warehouse structure version that is going to be created.

A.3 The dialog box Warehouse Schema Versioning

(II-1) Make a connection to a user warehouse by clicking  , the structure of a warehouse with schema versioning is tree-viewed in (II-1).

(II-2) Users provide our system with a textual decription of a NIAM/ORM conceptual schema modeling warehouse schema evolutions at the conceptual level. The description might be typed interactively in the text box (II-2) or might be automatically loaded by clicking the button  to open the dialog box OpenFileDialog for an available text file containing the description.

The following are actions on this dialog box to design a warehouse schema from a evolutions at the conceptual schema of a current warehouse version.

- Save the input in (II-2) into a text file: click
- Invoke the lexical analyzer and the parser: click
- Check the result from the lexical analyzing and parsing process: click
- Determine which kind of warehouse schema evolutions that include the addition/removal of a dimension and the addition/removal of a dimension level if the previous action is successfully accomplished: click . After the schema evolution is determined, generate a new warehouse schema of a new version from the warehouse schema of a current version and the schema evolution. The warehouse schema of the current version is then updated about the valid time.

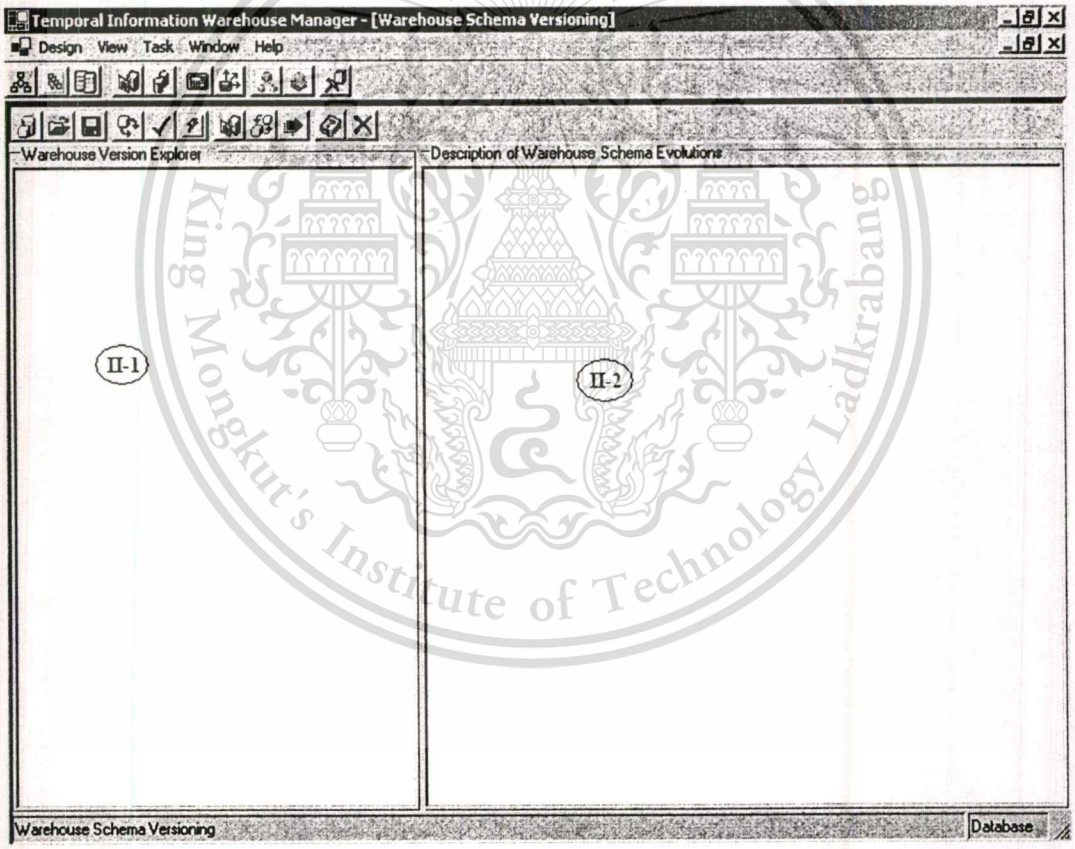


Figure A.3 The Dialog Box Warehouse Schema Versioning

- Save the information of the output warehouse schema of a new version and the affected information of the warehouse schema of a current version into 5NF meta tables: click

- Generate warehouse table structures belonging to a new version and consisting of fact structures and dimension structures with their primary keys handled by users: click 
- Generate warehouse table structures belonging to a new version and consisting of fact structures and dimension structures with primary keys automatically handled by our system for inter-version transformations: click  . These primary keys are obtained from user primary keys and added the information about the warehouse structure version that is going to be created.

A.4 The dialog box Design Mappings from an Operation Database to a Warehouse Version

(III-1) The output operational relational schema is tree-viewed and presented in DDL.

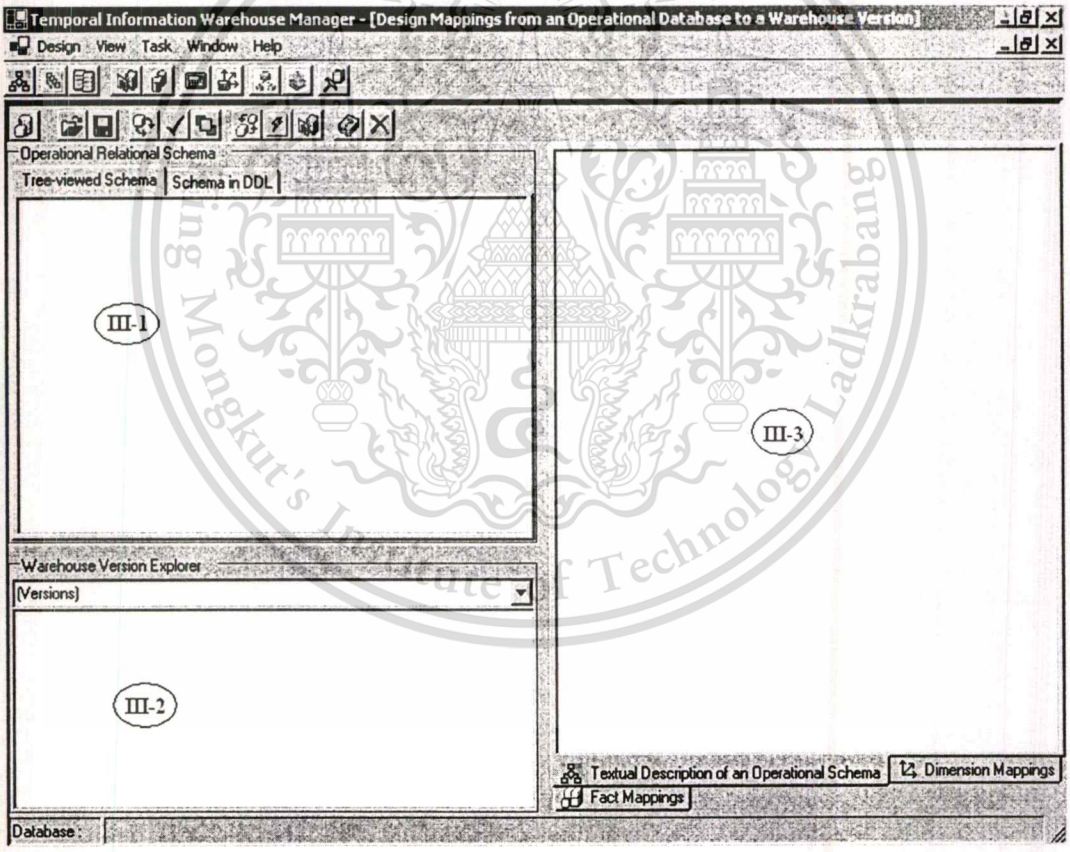


Figure A4 The Dialog Box for Meta of an Operational Database

(III-2) Make a connection to a user warehouse by clicking  , the structure of a warehouse with schema versioning is tree-viewed in (III-2).

(III-3) Users provide our system with a textual decription of a NIAM/ORM conceptual schema modeling an operational database. The description might be typed interactively in the text box (III-3) or might be automatically loaded by clicking the button to open the dialog box OpenFileDialog for an available text file containing the description.

The following are actions on this dialog box to obtain an operational relational schema from a conceptual schema of an operational database by applying the ONF algorithm.

- Save the input in (III-3) into a text file: click
- Invoke the lexical analyzer and the parser: click
- Check the result from the lexical analyzing and parsing process: click
- Carry out the ONF algorithm if the previous action is successfully accomplished: click
- Set up environmental parameters for the establishment of source-to-warheouse version mappings: click

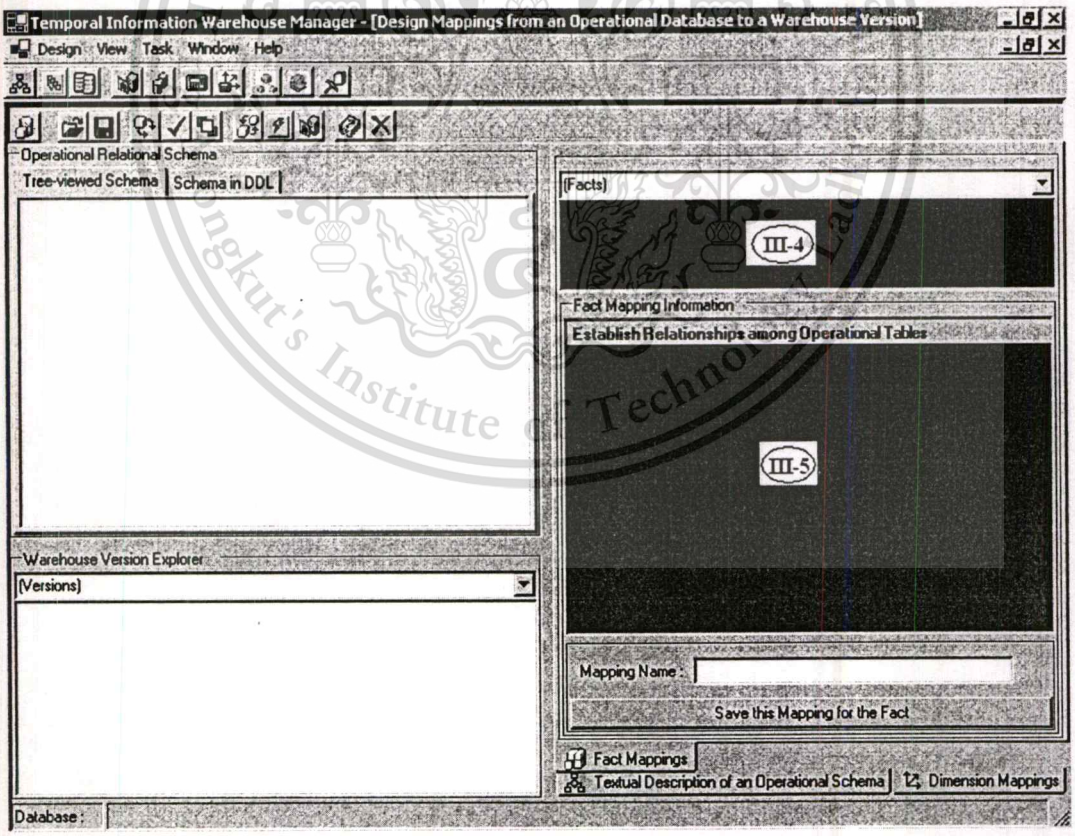


Figure A.5 The Dialog Box for a Fact Mapping

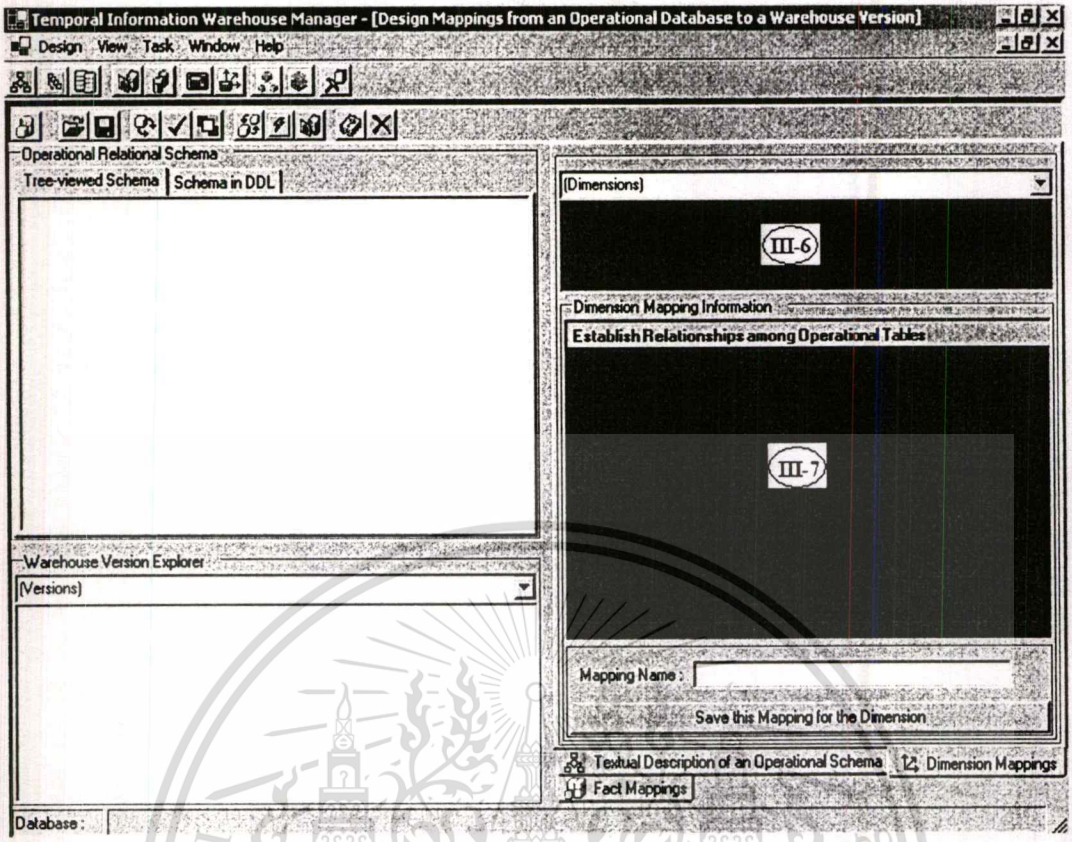


Figure A.6 The Dialog Box for a Dimension Mapping

- Save the information of the output operational relational schema corresponding to a selected version into 5NF meta tables if the previous action is successfully accomplished: click or click

(III-4) Establish column-to-column mappings of a fact table of a selected version. These mappings may require more than one table from an operational database. Therefore, the relationships among operational tables may be needed to compose a single table for data loading.

(III-5) Establish relationships among operational tables.

A mapping name for the fact mapping is requested for the management of the ETL process afterward. This fact mapping also needs to be saved.

Similar to fact mappings, dimension mappings are established in Figure A.6.

A.5 The dialog box Extraction, Transformation, and Loading Processing

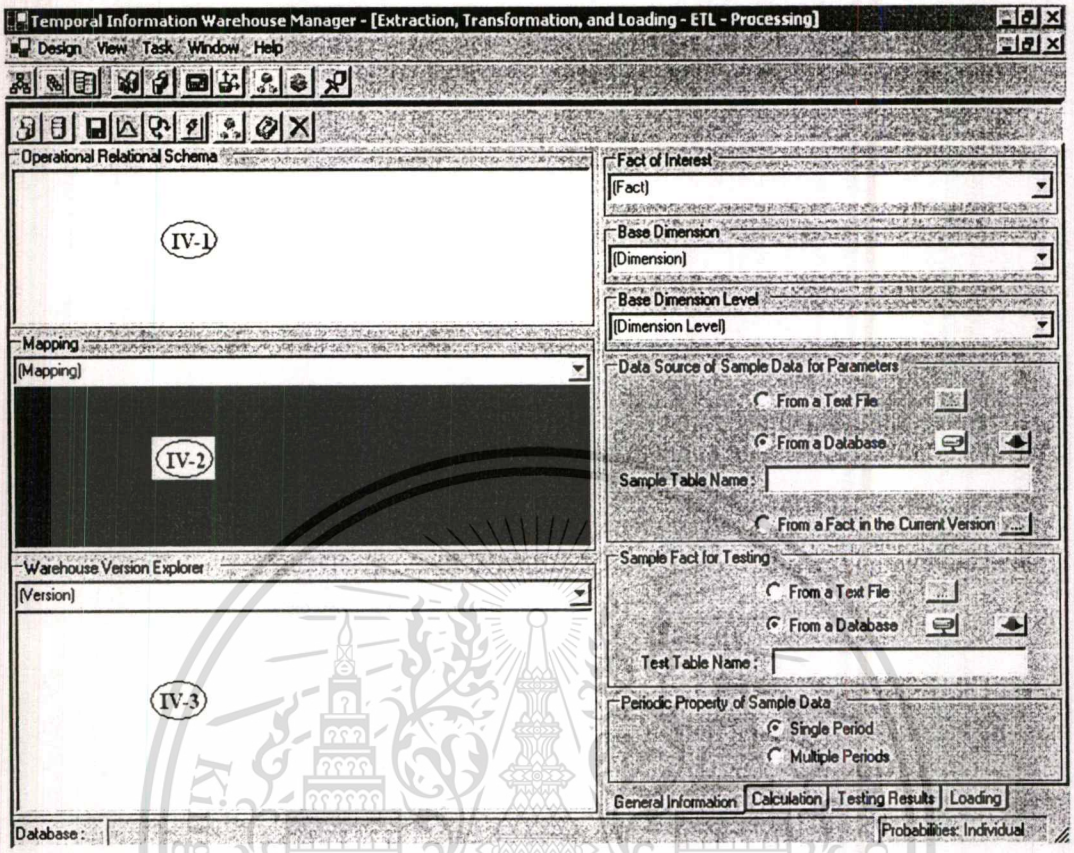


Figure A.7 The Dialog Box ETL Processing for Asking Users for General Information

(IV-1) Make a connection to an operational database from which data will be loaded into a selected warehouse version by clicking , the structure of the operational database is tree-viewed in (IV-1). Also all mappings including fact mappings and dimension mappings that have established in Figure A.5 and A.6 are retrieved and shown in (IV-2).

(IV-3) Make a connection to a user warehouse by clicking , the structure of the warehouse with schema versioning is tree-viewed in (IV-3).

(IV-4) is a space for informing users of errors that might occur during the loading process.

(IV-5) includes all data tables that are successfully processed for the filling of missing data and are able to be loaded from the chosen operational database. To make a connection to the operational database, the Data Link Properties dialog box in Figure 7.1 is invoked by clicking the button "Establish a Connection to the Operational Source". For the treatment of missing data, users are asked to choose a fact of their interest, a

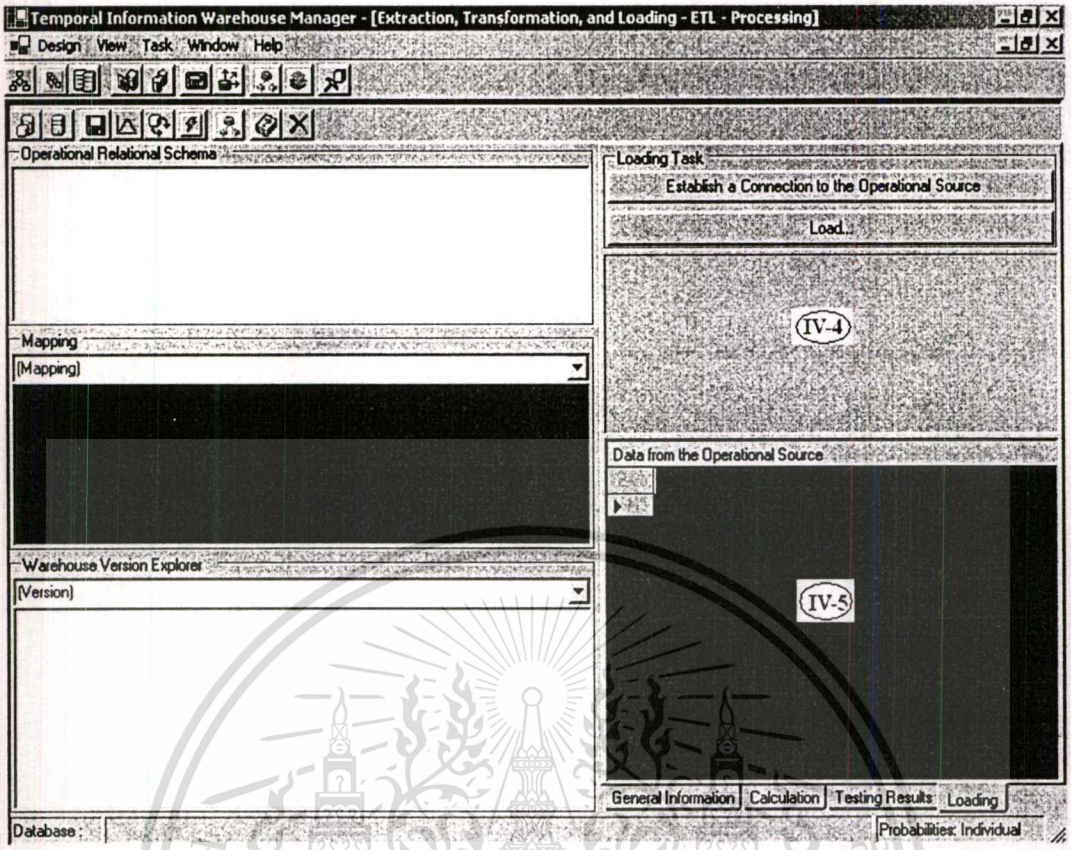


Figure A.8 The Dialog Box ETL Processing for Loading Operational Data

base dimension, a base dimension level belonging to the selected dimension. Besides, they are asked for data samples that may be loaded from a text file or from a data table in some database. This abpage "General Information" is used for the calculation of probabilities associated with a fact. The computing process is invoked by clicking . After the computation, click to save proportion values into 5NF meta tables for the handling of missing measure data in the corresponding fact.

is for the activation of the tabpage "Loading" for the loading process.

A.6 The dialog box Inter-Version Transforming

(V-1) Make a connection to a user warehouse by clicking , the structure of the warehouse with schema versioning is tree-viewed in (V-1). Along with the structure of the warehouse, schema evolutions during the life span of the warehouse are retrieved and shown in (V-2). The information on schema evolutions will be useful for the establishment of inter-version transformations.

Users are asked to specify a pair of versions. They are adjacent to each other. The

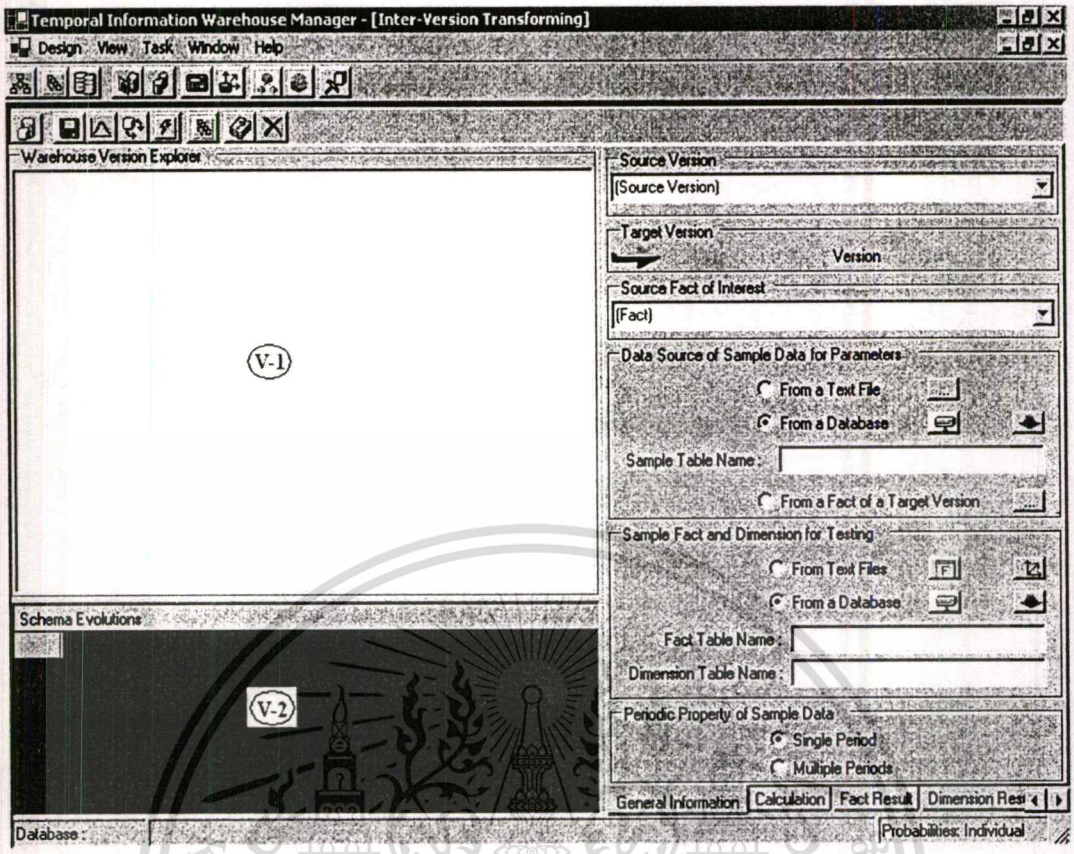


Figure A.9 The Dialog Box Inter-Version Transforming for General Information

process of setting up parameters for the inter-version transformation is similar to one for the handling of missing measure data in Figure A.7. It is activated by clicking .

The calculating process is activated by clicking .

After the computation of probabilities, these values are saved into meta tables for a corresponding fact in a selected version by clicking .

The inter-version transformation may be tested in advance with probabilities which have been obtained by test data provided by users and clicking .

Finally, the transformation process is activated by clicking .

The transformation process might be carried out with at least one involved fact in the source version by selecting them in the CheckedListBox (V-3). All data of source dimensions and source facts are pre-viewed in (V-4). The output data after the transformation are viewed in (V-5). Not only measure data, but also affected dimension data are transformed.

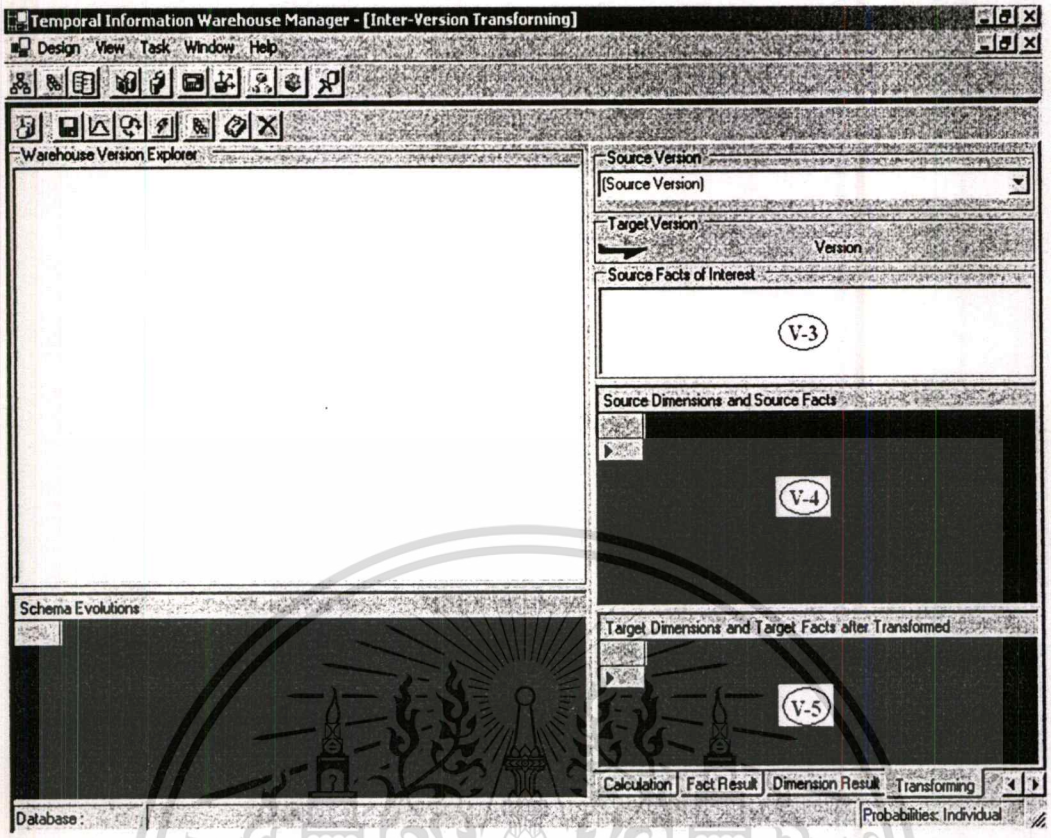


Figure A.10 The Dialog Box Inter-Version Transforming for Transforming Data

A.7 The dialog box Querying_Analyzing

(VI-1) Make a connection to a user warehouse by clicking , the structure of the warehouse with schema versioning is tree-viewed in (VI-1).

(VI-3) is a text box that receives string queries in SQL for "select" commands. Click to execute the submitted "select" command in the tabpage "SQL Querying" in Figure A.11. The respective result will be given in (VI-4).

Click to activate the tabpage "Reporting" in Figure A.12.

Click to activate the tabpage "OLAP" in Figure A.13.

Click to activate the tabpage "Data Mining" in Figure A.14.

(VI-2) is a datagrid to contain data tables belonging to a selected version that is called a base version for the current querying and analyzing process.

Users are asked for the selection of a fact and a dimension associated with the selected fact so that a new table produced from the join of them. This table will be provided for the Microsoft Privot Table in (VI-5). From the Pivot Table services, an interaction with

Microsoft Excel is able to be performed and reports are then easily made with functionality of Microsoft Excel.

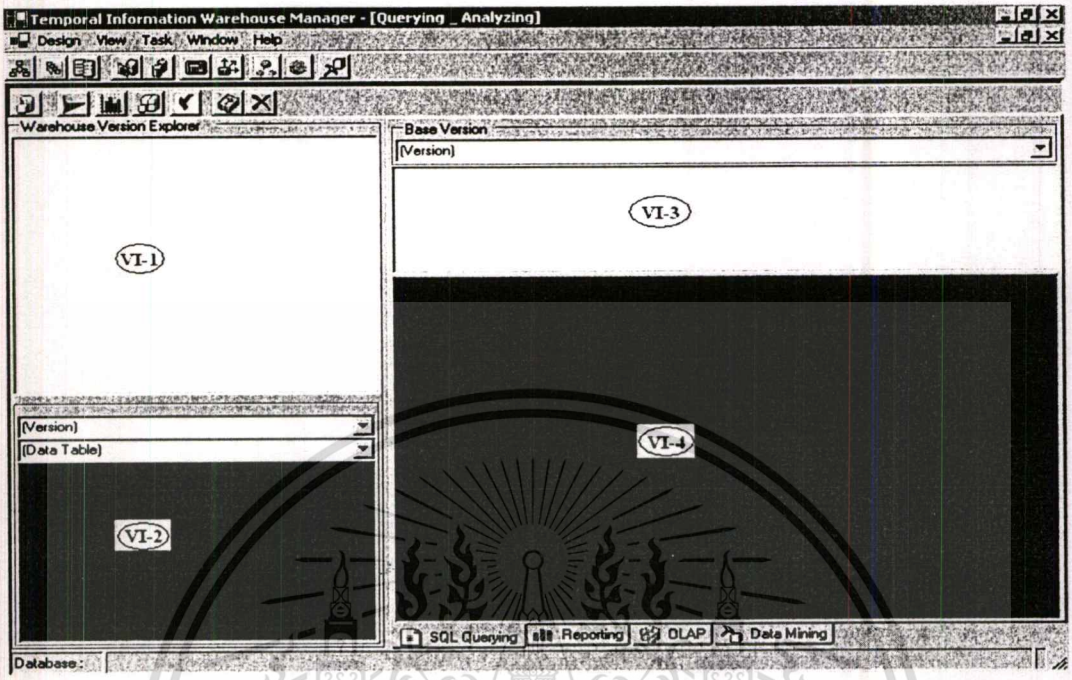


Figure A.11 The Dialog Box Querying_Analyzing for SQL Querying

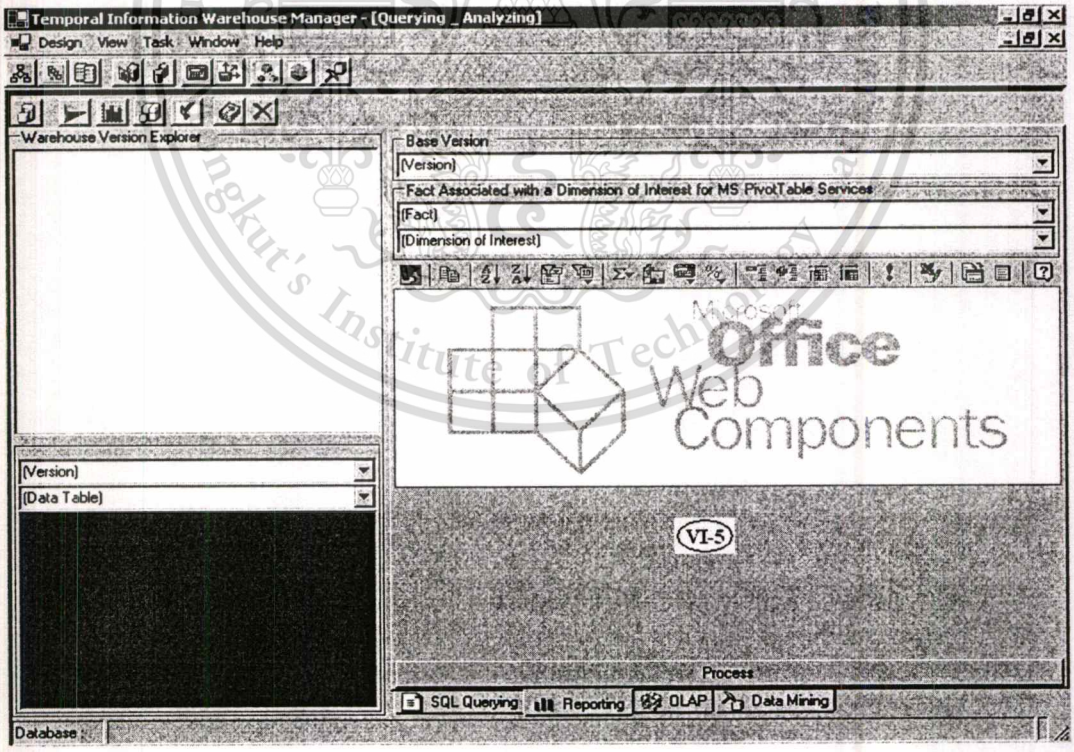


Figure A12 The Dialog Box Querying_Analyzing for Reporting with MS PivotTable Services

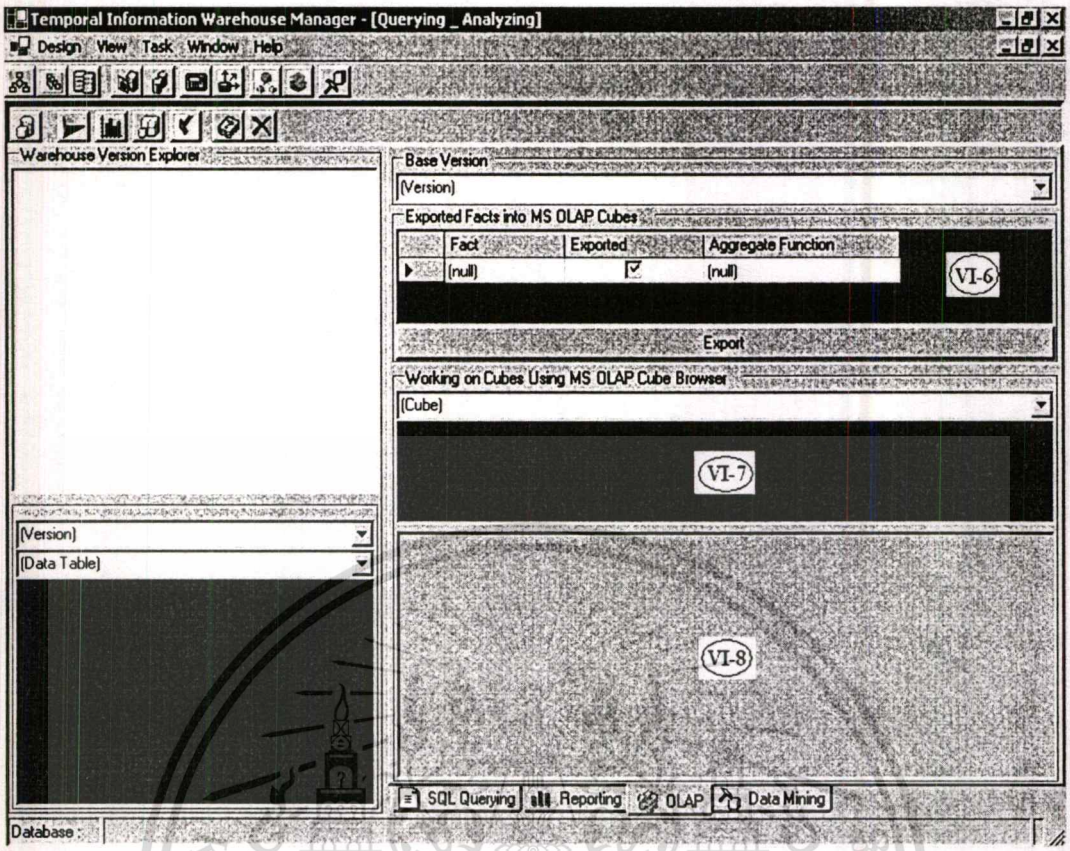


Figure A.13 The Dialog Box Querying_Analyzing for Analysis Services using Cubes

(VI-6) requires users to specify facts which are going to be exported into OLAP cubes managed by Microsoft OLAP Manager Cube Browser. From our design of a fact structure with a measure attribute, it is convenient to play with cubes at this stage. Also, an aggregate function associated with a new cube is necessary to be defined for the establishment of parameters of the Microsoft OLAP cube browser. After the exporting of temporal warehouse data to OLAP cubes, these new cubes are able to be analyzed in (VI-7) and (VI-8). Once cubes are generated, users may manipulate them directly using Microsoft SQL Server 2000 Analysis Services.

As for data mining, users will provide our system in (VI-9) with a base version, a case dimension for data mining algorithms and determine one of two available data mining algorithms supported by Microsoft Data Mining Model Browser. After the creation of a data mining model, the model is able to be directly processed using Microsof t SQL Server 2000 Analysis Services or using the Data Mining Modeling Browser that is employed in our system in (VI-10).

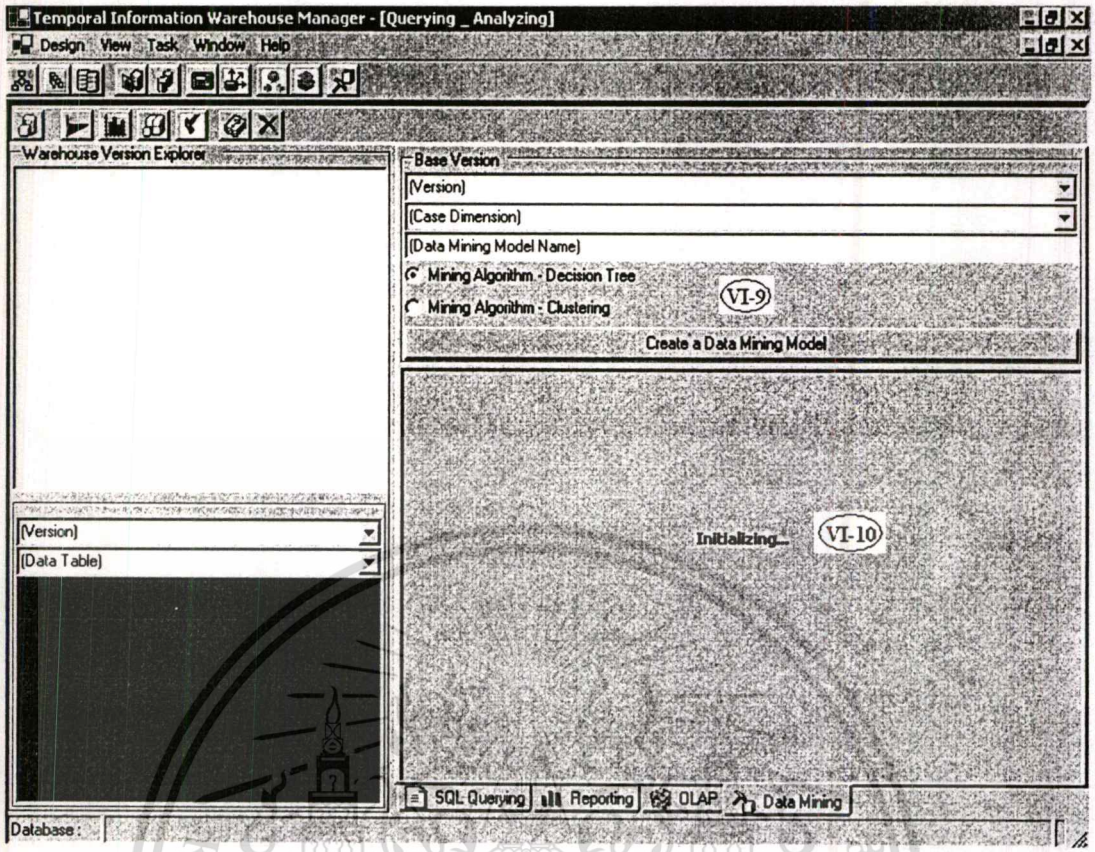


Figure A.14 The Dialog Box Querying_Analyzing for Data Mining

Appendix B

Case Study 1

This case study is designed from an example on page # 220 in [56]. The output temporal information warehouse is 'chau37' using Microsoft SQL Server 2000. Steps for the development of the warehouse are illustrated through figures.

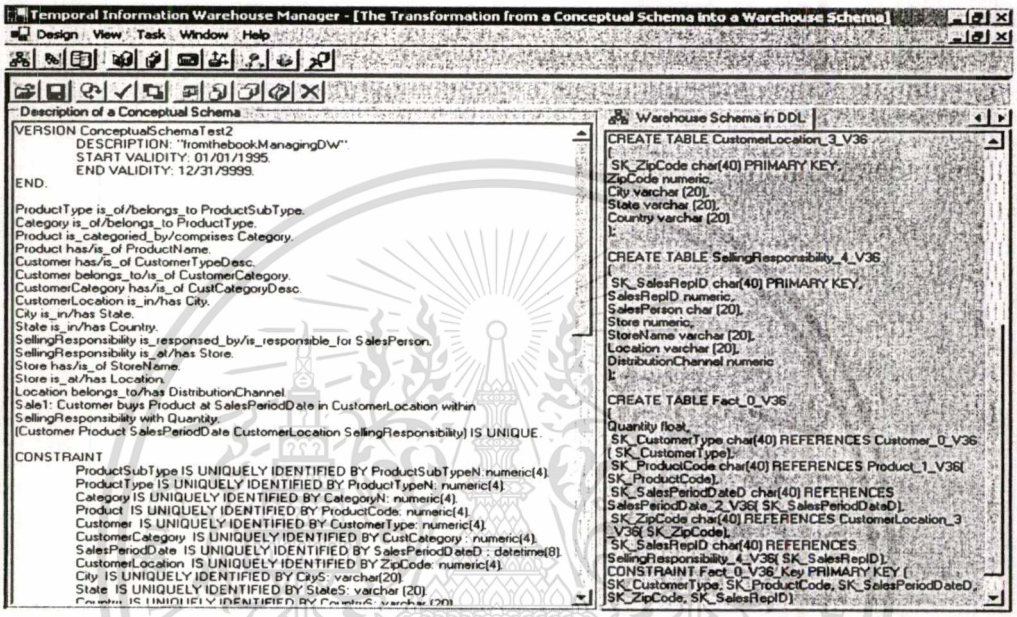


Figure B.1 The Design of a Star Schema for the First Version in a Warehouse chau37

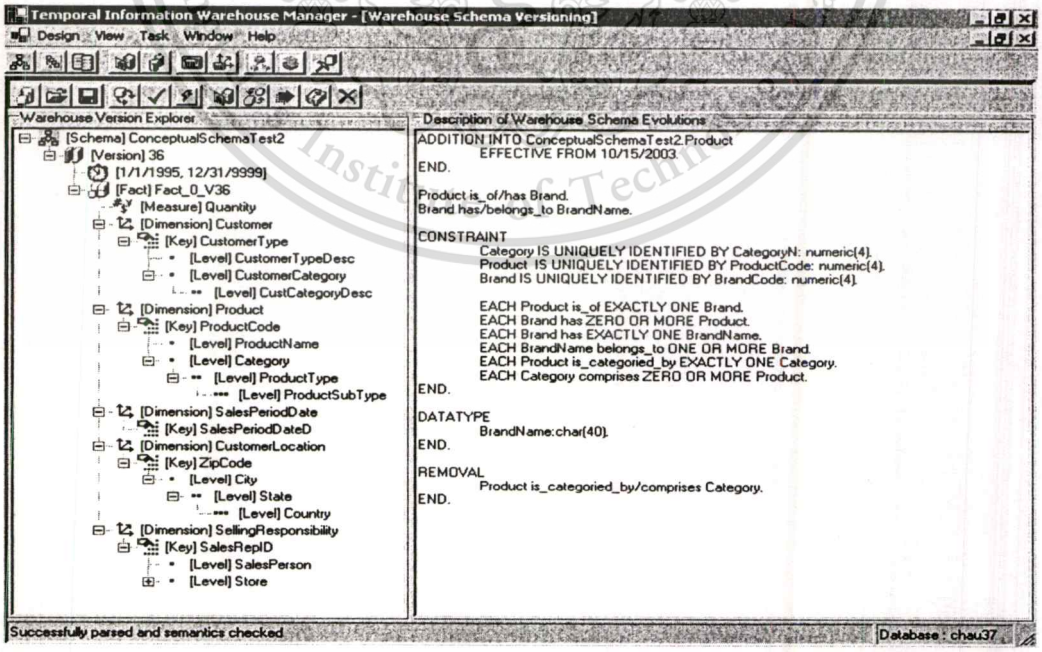


Figure B.2 The Warehouse Structure before the Addition of the Level Brand into Product

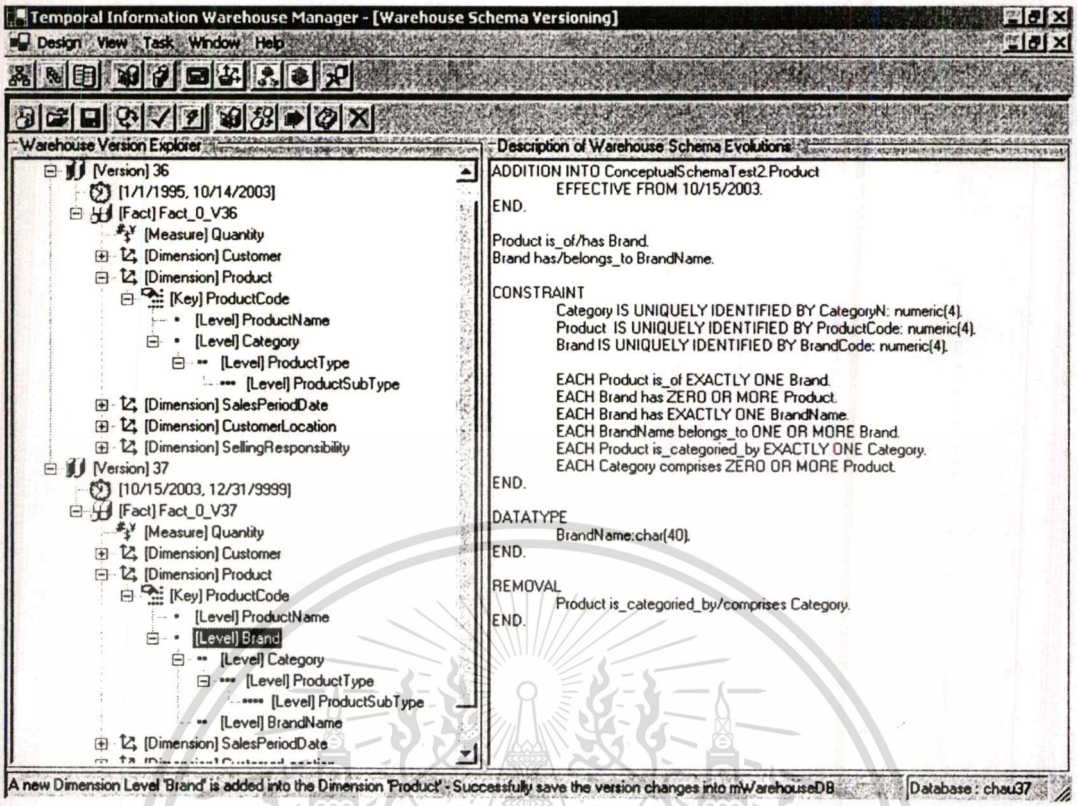


Figure B.3 The Structure of the Warehouse after Schema Evolving

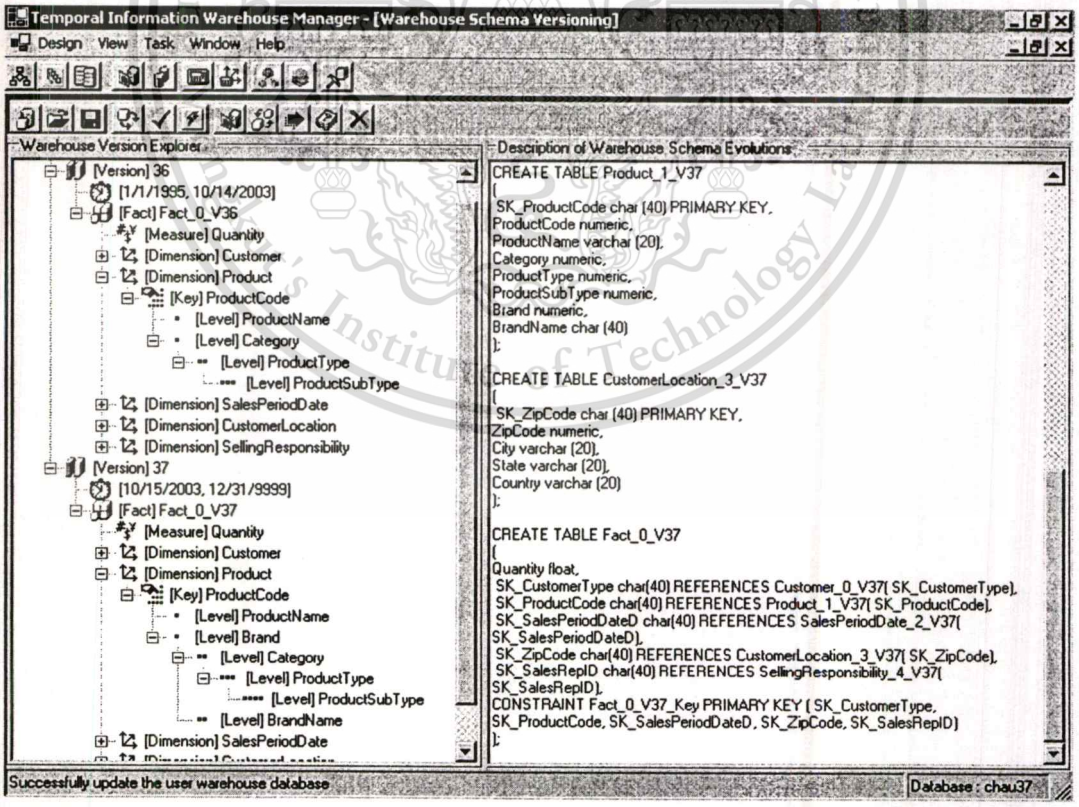


Figure B.4 The Generation of Table Structures of a New Version in the Warehouse

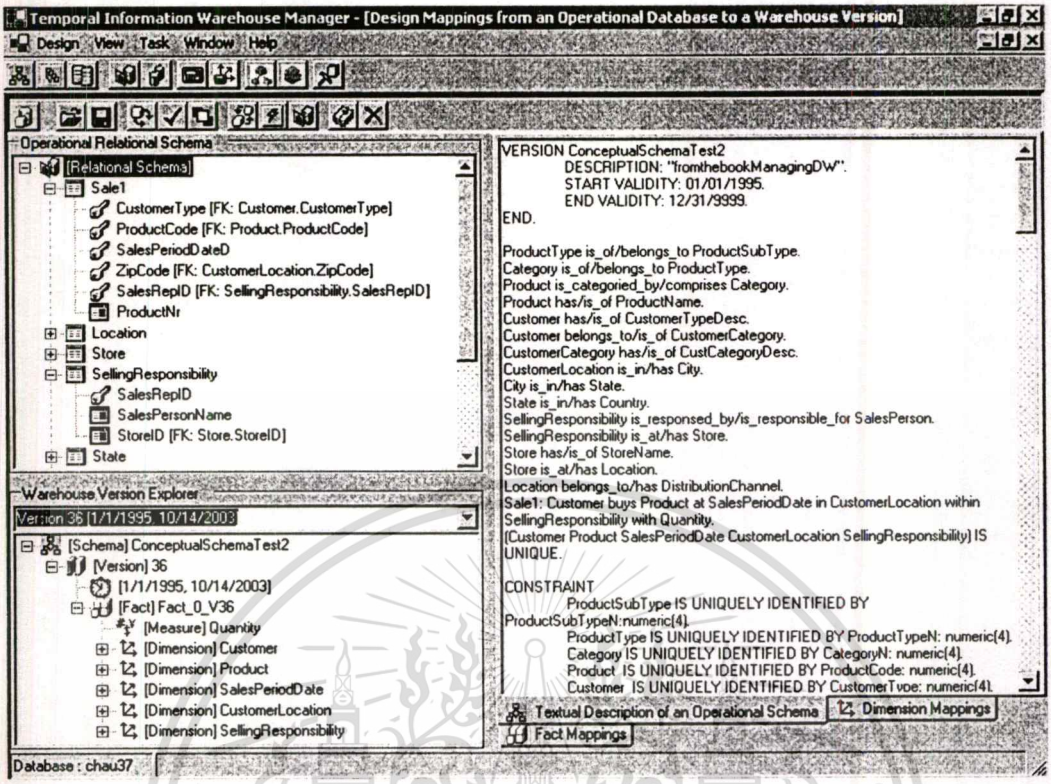


Figure B.5 The Gathering of Meta Data of an Operational Database for the Version 36

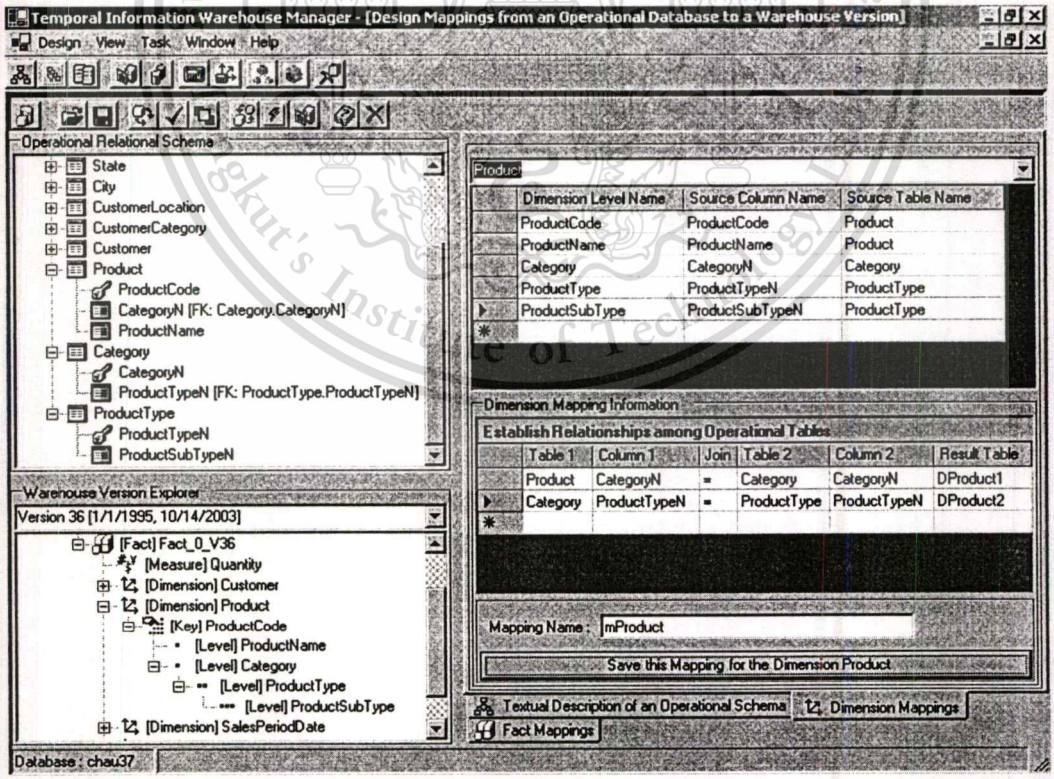


Figure B.6 A Product Mapping

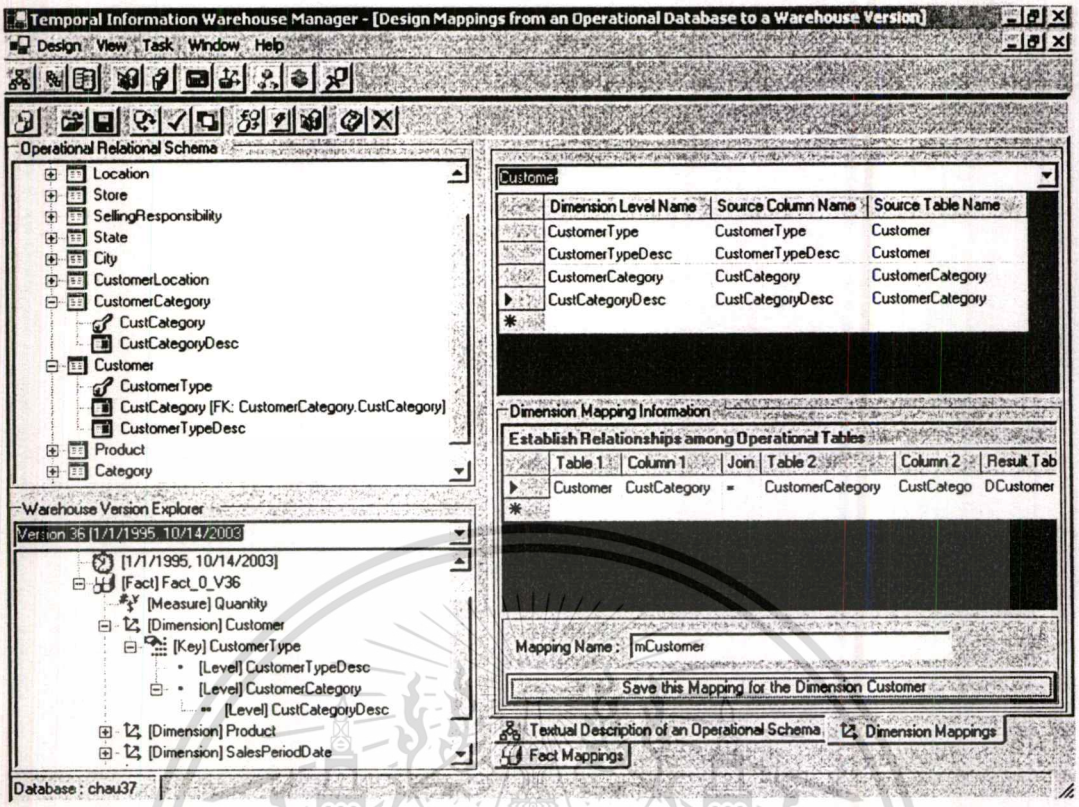


Figure B.7 A Customer Mapping

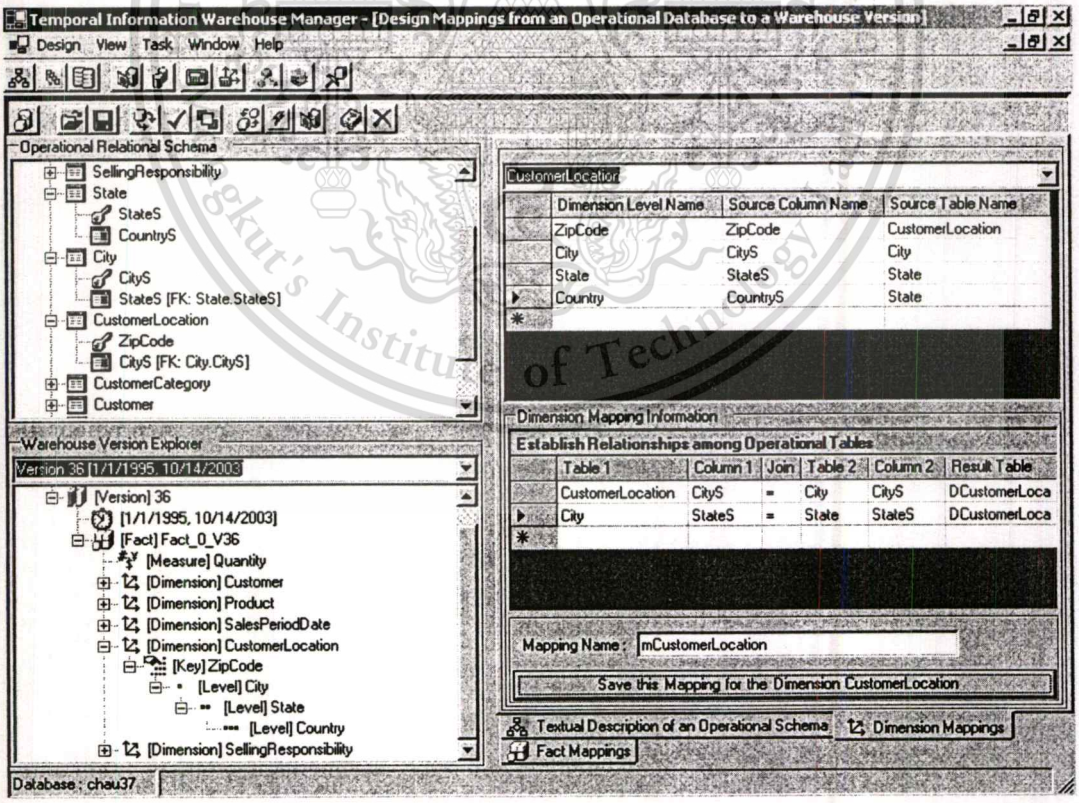


Figure B.8 A CustomerLocation Mapping

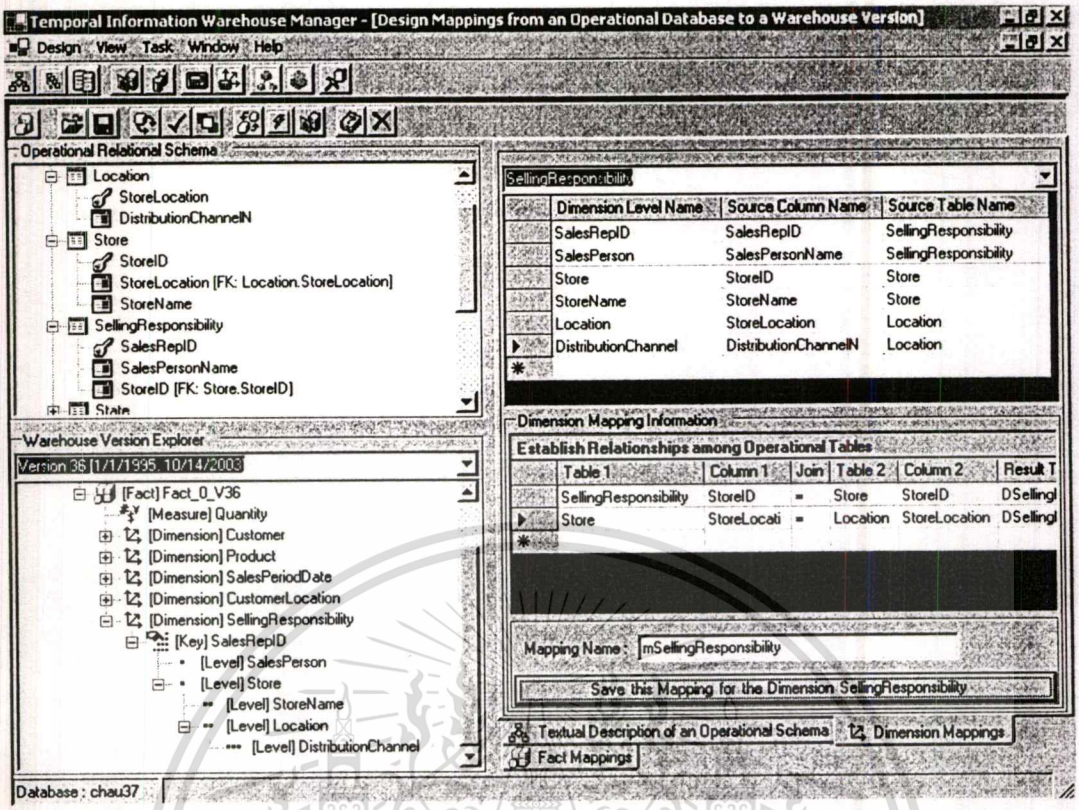


Figure B.9 A SellingResponsibility Mapping

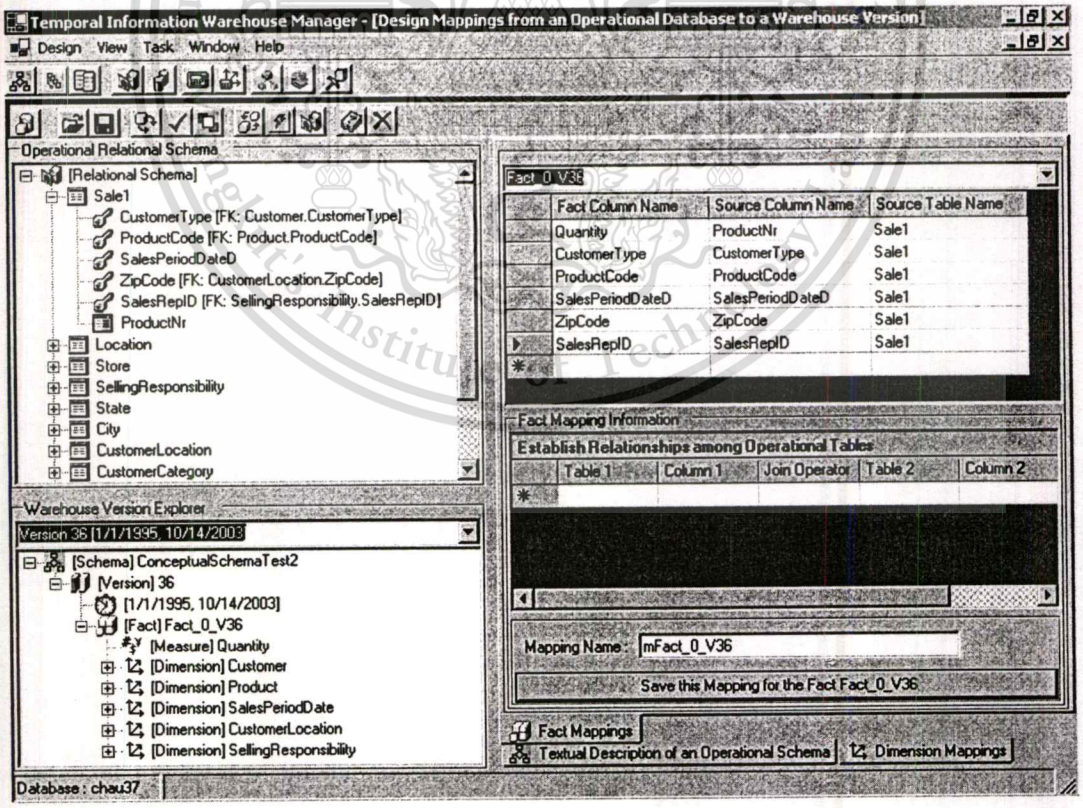


Figure B.10 A Fact_0_V36 Mapping

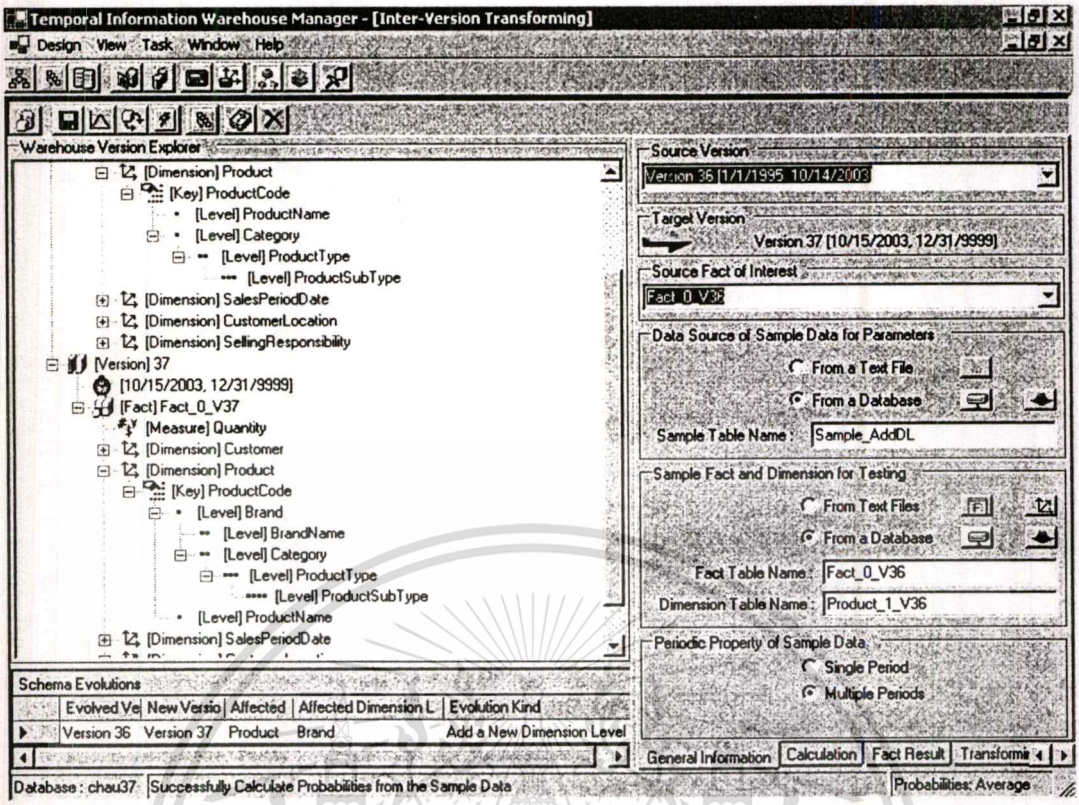


Figure B.11 The Establishment of Parameters for the Proportion Computation

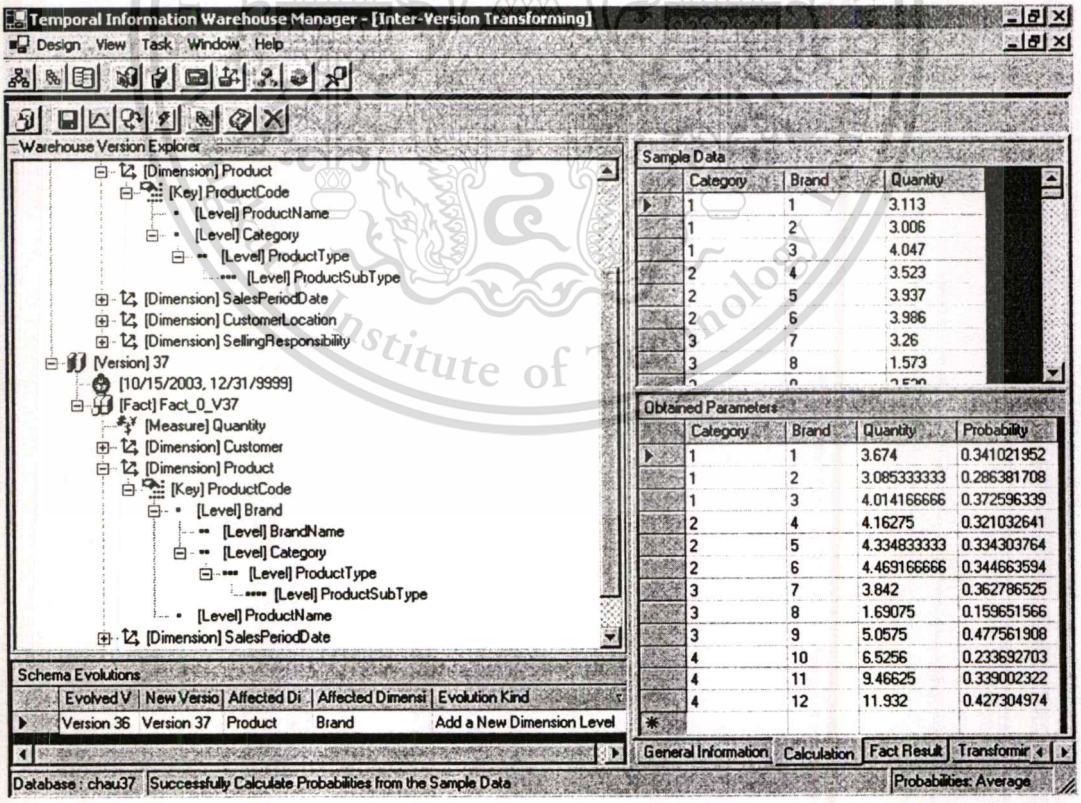


Figure B.12 Obtained Probabilities

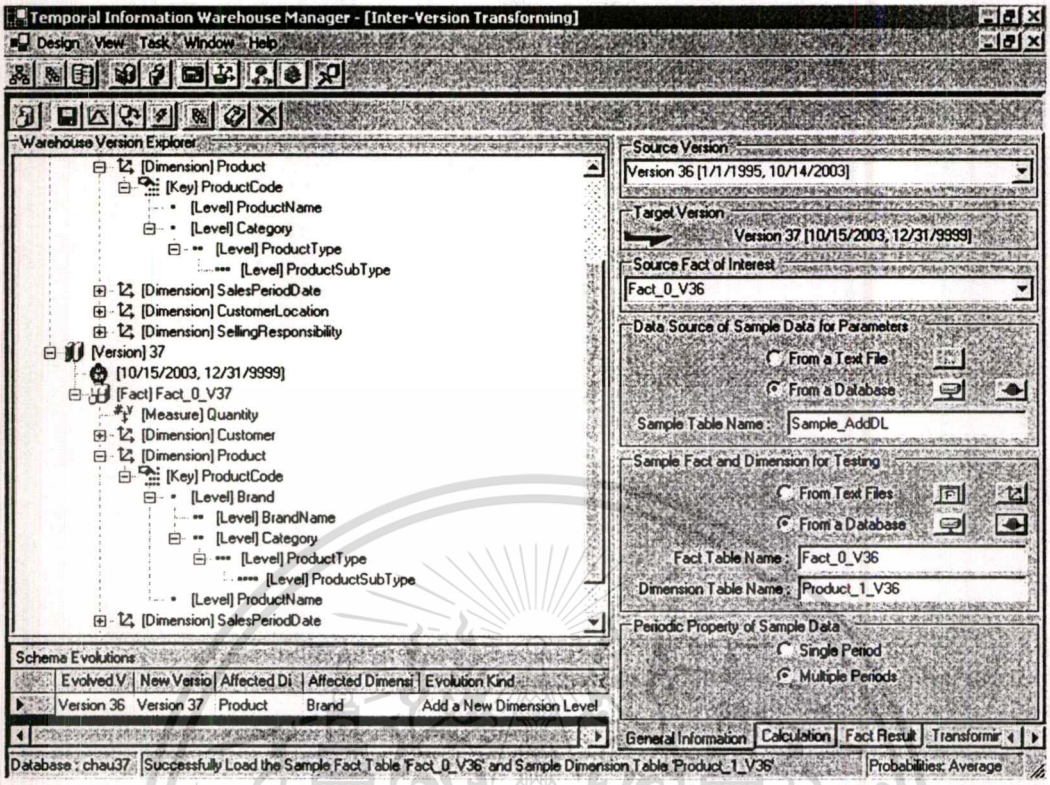


Figure B.13 The Loading of a Sample Fact and Dimension Tables for Testing

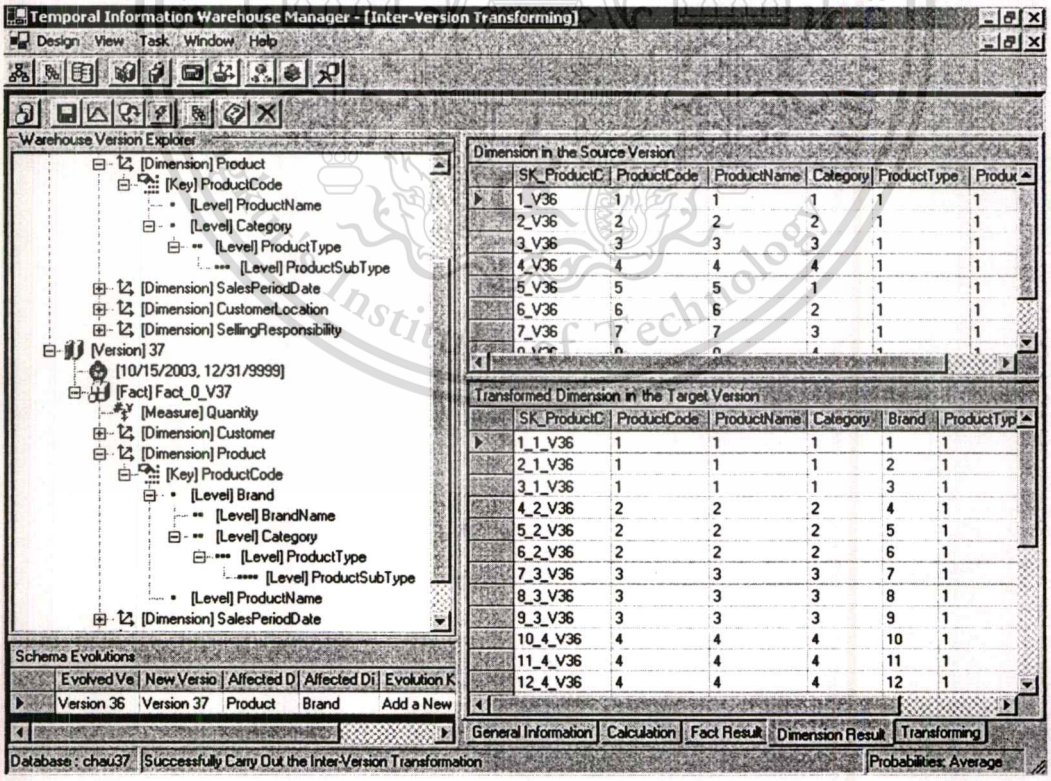


Figure B.14 Transformed Dimension Data

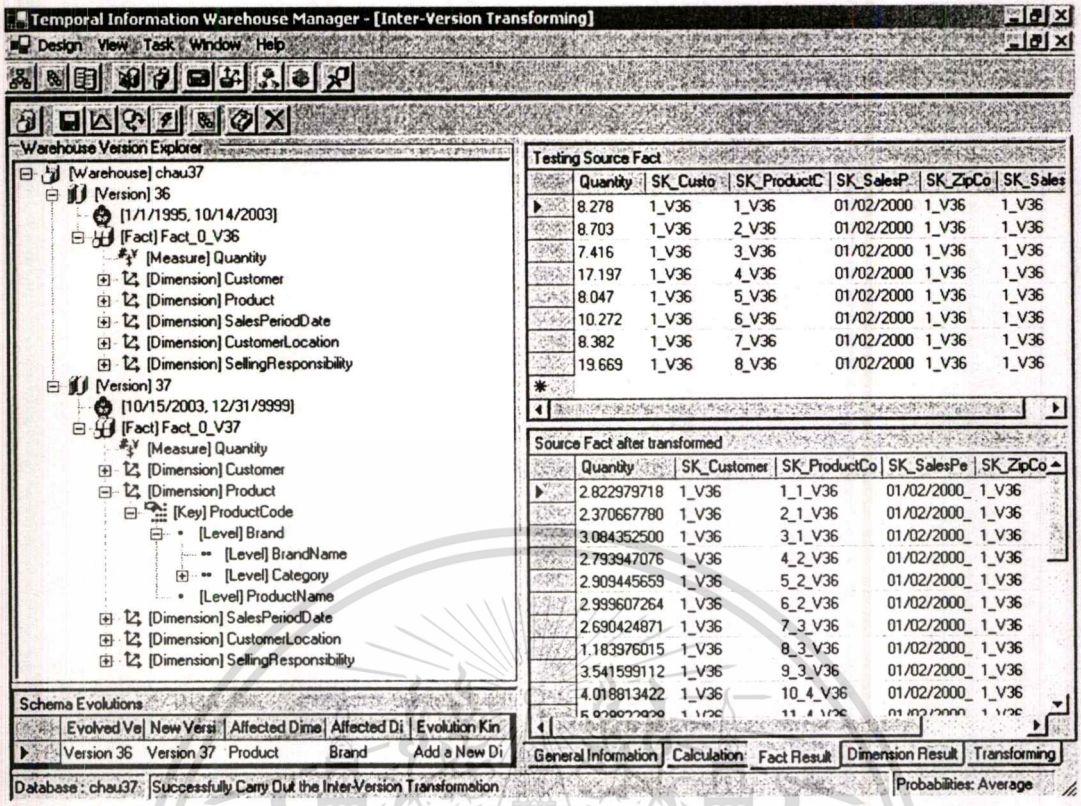


Figure B.15 Transformed Fact Data

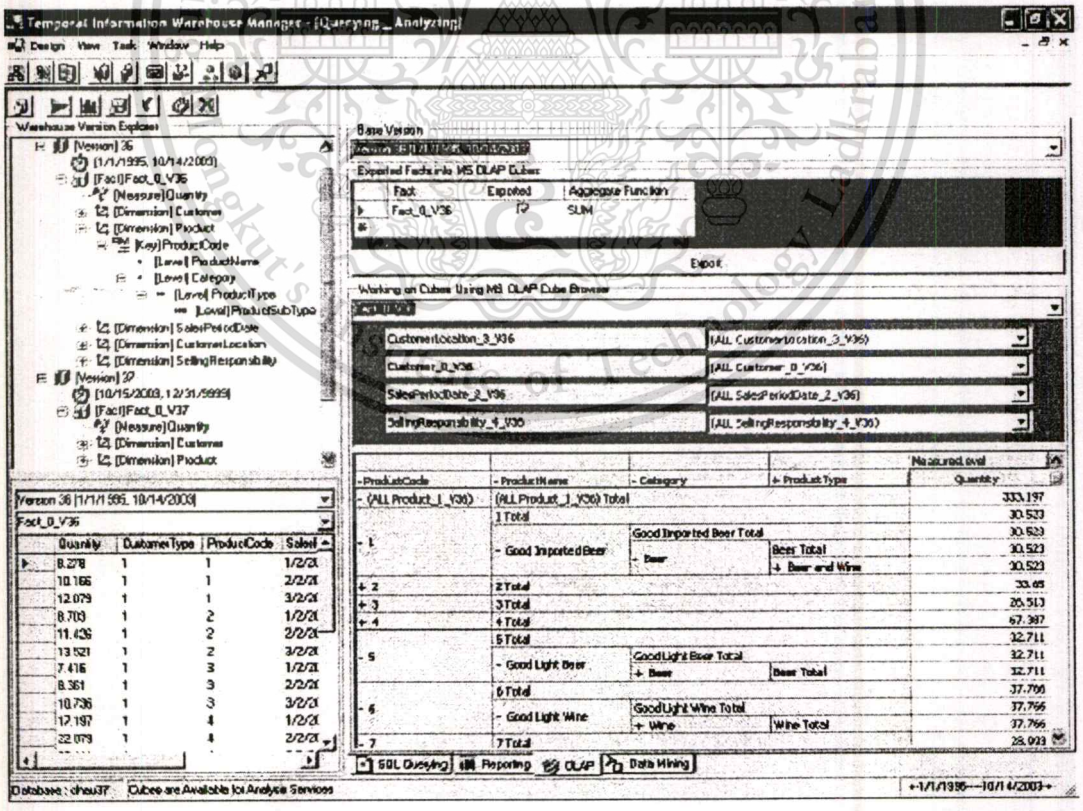


Figure B.16 The Manipulation on a New Cube with the Aggregate Function SUM

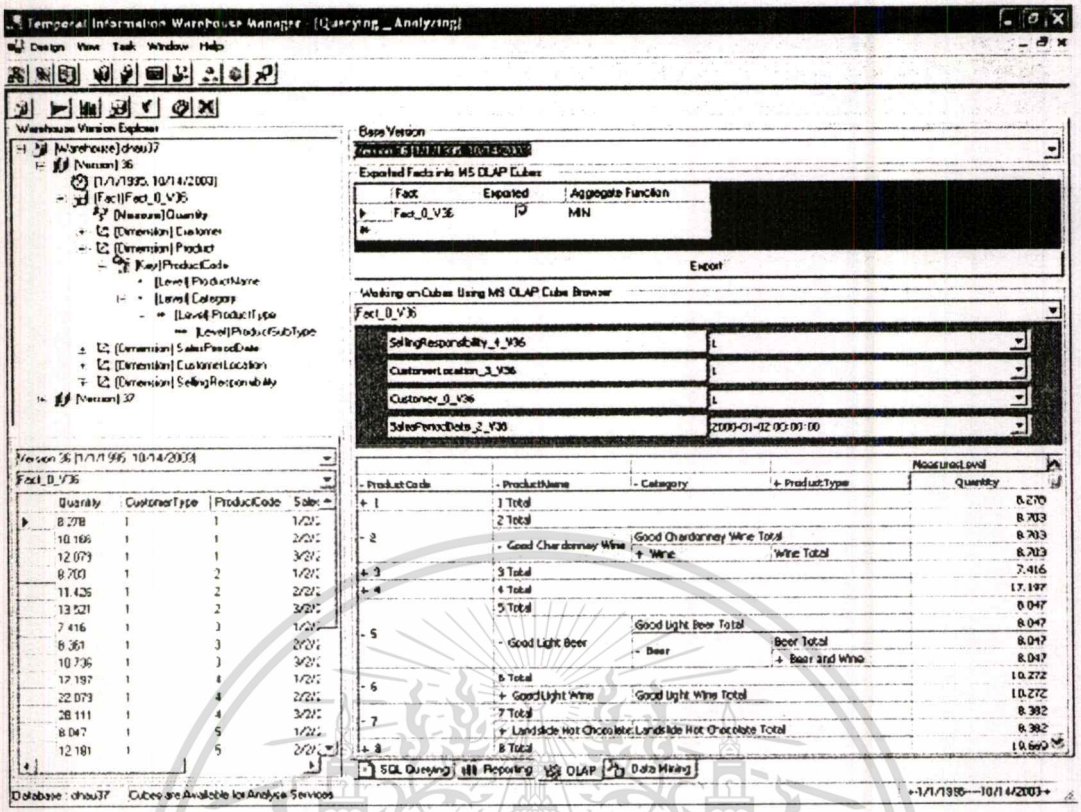


Figure B.17 The Manipulation on a New Cube with the Aggregate Function MIN

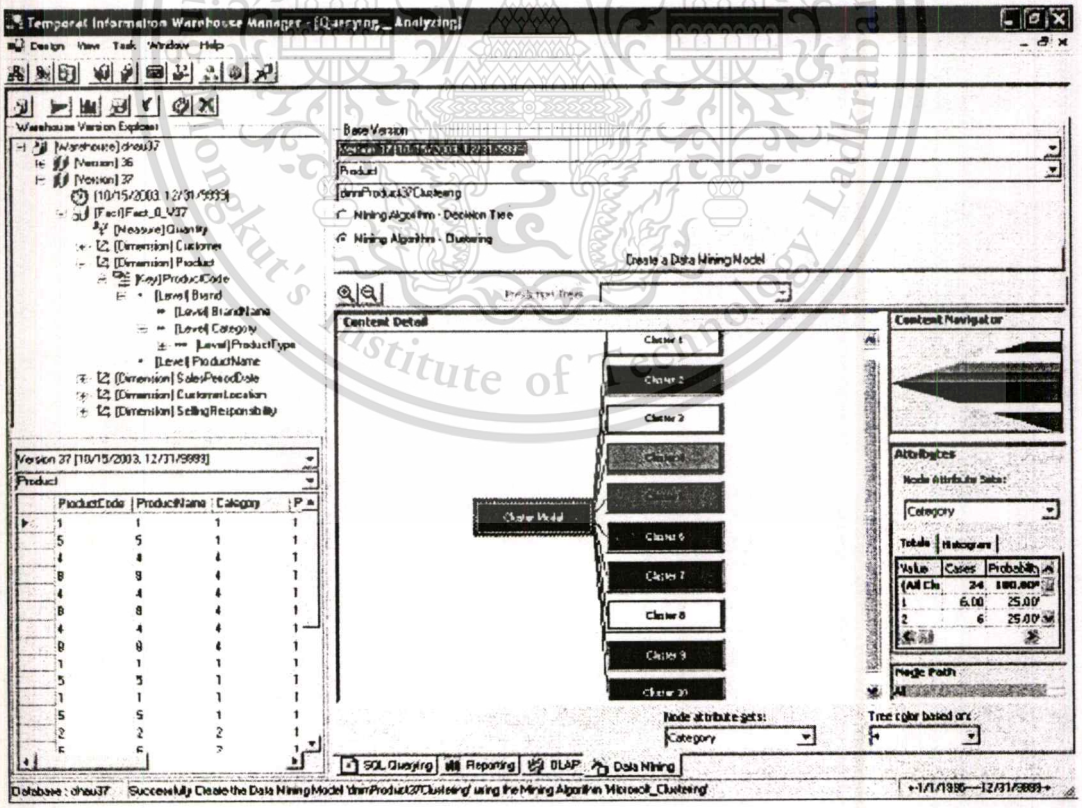


Figure B.18 The Data Mining Algorithm – Clustering – on the Case Dimension Product

Appendix C

Calculation Examples in the Missing Data Process

This section provides a short description of how to calculate proportions from a sample data set from table B.9 in [13] containing Monthly Champagne Sales in [Jan 1962, Dec 1969]. The data in 1965 is selected as a single period sample data set for the calculation of individual proportions according to the first choice in our proposed technique. The data in [1964, 1969] is selected as a 6-period sample data set for the calculation of average proportions according to the second choice. They are shown in Table C.1. A selected base dimension is the dimension Time and a selected base dimension level in the dimension Time is Year.

Table C. 1 Sample Data Set in [13]

	1964	1965	1966	1967	1968	1969
Jan	3.113	3.653	3.633	4.016	2.639	3.934
Feb	3.006	3.176	4.292	3.957	2.899	3.162
Mar	4.047	3.812	4.154	4.51	3.37	4.286
Apr	3.523	4.129	4.121	4.276	3.74	4.676
May	3.937	4.288	4.647	4.968	2.927	5.01
Jun	3.986	4.447	4.753	4.677	3.986	4.874
Jul	3.26	3.812	3.965	3.523	4.217	4.633
Aug	1.573	1.906	1.723	1.821	1.738	1.659
Sep	3.528	4.765	5.048	5.222	5.221	5.951
Oct	5.211	5.876	6.922	6.873	6.424	6.981
Nov	7.614	9.053	9.858	10.803	9.842	9.851
Dec	9.254	11.276	11.331	13.916	13.076	12.67

C.1 The Calculation of Individual Proportions in Table 4.7

Summation(1965) = 3.653 + 3.176 + 3.812 + 4.129 + 4.288 + 4.447 + 3.812 +
1.906 + 4.765 + 5.876 + 9.053 + 11.276 = 60.192

Proportion(Jan) = 3.653/Summation(1965) = 3.653/60.192 = 0.060686

Proportion(Feb) = 3.176/Summation(1965) = 3.176/60.192 = 0.0527704

Proportion(Mar) = 3.812/Summation(1965) = 3.812/60.192 = 0.0633245

Proportion(Apr) = 4.129 /Summation(1965) = 4.129/60.192 = 0.06860158

Proportion(May) = 4.288 /Summation(1965) = 4.288/60.192 = 0.07124010554

Proportion(Jun) = 4.447 /Summation(1965) = 4.447/60.192 = 0.0738786279

Proportion(Jul) = 3.812 /Summation(1965) = 3.812/60.192 = 0.063324538

Proportion(Aug) = 1.906 /Summation(1965) = 1.906/60.192 = 0.031662269

Proportion(Sep) = 4.765 /Summation(1965) = 4.765/60.192 = 0.07915567

Proportion(Oct) = 5.876 /Summation(1965) = 5.876/60.192 = 0.0976253298

Proportion(Nov) = 9.053 /Summation(1965) = 9.053/60.192 = 0.150395778

Proportion(Dec) = 11.276 /Summation(1965) = 11.276/60.192 = 0.18733509

C.2 The Calculation of Average Proportions in Table 4.8

Summation = m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12

Where:

m1 is obtained from the clustering on (3.113, 3.653, 3.633, 4.016, 2.639, 3.934);

m2 from the clustering on (3.006, 3.176, 4.292, 3.957, 2.899, 3.162);

m3 from the clustering on (4.047, 3.812, 4.154; 4.51, 3.37, 4.286);

m4 from the clustering on (3.523, 4.129, 4.121, 4.276, 3.74, 4.676);

m5 from the clustering on (3.937, 4.288, 4.647, 4.968, 2.927, 5.01);

m6 from the clustering on (3.986, 4.447, 4.753, 4.677, 3.986, 4.874);

m7 from the clustering on (3.26, 3.812, 3.965, 3.523, 4.217, 4.633);

m8 from the clustering on (1.573, 1.906, 1.723, 1.821, 1.738, 1.659);

m9 from the clustering on (3.528, 4.765, 5.048, 5.222, 5.221, 5.951);

m10 from the clustering on (5.211, 5.876, 6.922, 6.873, 6.424, 6.981);

m11 from the clustering on (7.614, 9.053, 9.858, 10.803, 9.842, 9.851);

m12 from the clustering on (9.254, 11.276, 11.331, 13.916, 13.076, 12.67).

Proportion(Jan) = m1/Summation = 0.0645161

Proportion(Feb) = m2/Summation = 0.04838709

Proportion(Mar) = m3/Summation = 0.0645161

Proportion(Apr) = m4/Summation = 0.0645161

Proportion(May) = m5/Summation = 0.0645161

Proportion(Jun) = m6/Summation = 0.0645161

Proportion(Jul) = m7/Summation = 0.0645161

Proportion(Aug) = m8/Summation = 0.03225806

Proportion(Sep) = m9/Summation = 0.080645161

Proportion(Oct) = m10/Summation = 0.1129302258

Proportion(Nov) = m11/Summation = 0.14516129

Proportion(Dec) = m12/Summation = 0.19354838

C.3 The Filling of Missing Data in Figure 4.11

- Using Individual Proportions

- For the missing value at the position with TKey = 3:

  <NULL3> = 2.851*Proportion(Mar)/Proportion(Jan) = 2.851*0.0633245/0.060686

  = 2.975

- For the missing value at the position with TKey = 6:

  <NULL6> = 2.851*Proportion(Jun)/Proportion(Jan) = 2.851*0.0738786/0.060686

  = 3.4708

- For the missing value at the position with TKey = 8:

  <NULL8> = 2.851*Proportion(Aug)/Proportion(Jan) = 2.851*0.0316622/0.060686

  = 1.4875

and so on for other missing positions.

- Using Average Proportions

- For the missing value at the position with TKey = 3:

  <NULL3> = 2.851*Proportion(Mar)/Proportion(Jan) = 2.851*0.0645161/0.064516

  = 2.851

- For the missing value at the position with TKey = 6:

  <NULL6> = 2.851*Proportion(Jun)/Proportion(Jan) = 2.851*0.0645161/0.064516

  = 2.851

- For the missing value at the position with TKey = 8:

  <NULL8> = 2.851*Proportion(Aug)/Proportion(Jan) = 2.851*0.0322580/0.064516

  = 1.4255

and so on for other missing positions.

Author Biography

Miss. Vo Thi Ngoc Chau graduated from Ho Chi Minh University of Technology, Vietnam after fulfilling four and a half years for the Bachelors' Degree in Computer Engineering. She was placed at the fifth rank out of the class of 159 students at the Faculty of Information Technology in 2003. She was awarded a scholarship from the ASEAN University Network / Southeast Asia Engineering Education Development Network (AUN/SEED-NET) Project for the Master's degree at the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Thailand, in May 2003. Her research interests include database systems, information system analysis and design, knowledge management, data mining techniques, and artificial intelligent techniques.