

**DIMENSION UPDATES IN DATA WAREHOUSES**

**By**

**HAIHONG MA**

**Bachelor of Science**

**Dalian University of Technology**

**Dalian, P. R. China**

**1992**

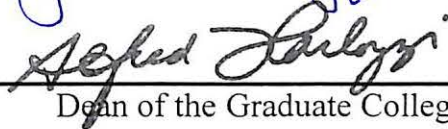
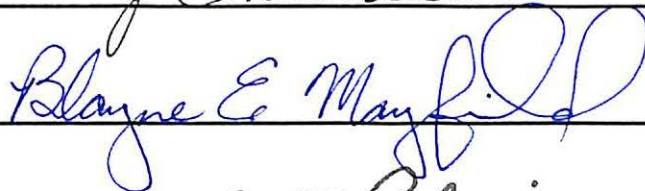
**Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
In partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
December 2004**

DIMENSION UPDATES IN DATA WAREHOUSES

Thesis Approved:



Thesis Advisor



Dean of the Graduate College

## ACKNOWLEDGEMENTS

I would like to express my appreciation to my advisor, Dr. G. E. Hedrick for his guidance and understanding. Without his help, I would not have finished my thesis smoothly and timely. Great thanks are also to my other committee members: Dr. J. P. Chandler and Dr. B. Mayfield, whose guidance, assistance, encouragement, and friendship were invaluable.

Thanks to Computer Science Department, which gave me a chance to study and earn my master's degree. Also thanks to all the faculty members both in Tulsa and Stillwater campus who have helped me in this period.

Special thanks to my parents and my sisters for their love and support. And I would like to devote this piece of work to my daughter Grace Xu and my husband Dapeng Xu, the two persons I love deeply.

## TABLE OF CONTENTS

Chapter		Page
1	Introduction .....	1
	1.1 Data Warehouse Concepts .....	1
	1.2 Data Warehouse Refreshment .....	2
2	Dimensional Data Model .....	5
	2.1 Entity Relationship Model vs. Dimensional Model .....	5
	2.2 Star Schema .....	7
3	Related Work .....	10
4	Dimension Update Model .....	12
	4.1 Dataflow in Data Warehouse .....	12
	4.2 Dimension Maintenance .....	14
	4.3 Dimension Schema & Instance .....	16
	4.4 Dimension Storage .....	18
	4.5 Dimension Operations .....	19
	4.6 Dimension Consistency .....	20
5	Dimension Update Strategy .....	22
	5.1 Motivation .....	22
	5.2 Dimension Update Algorithm .....	24
6	Dimension Update Implementation .....	29
	6.1 Dimension Update Model Simulation .....	29
	6.2 Experiments and Performance .....	30
7	Conclusions and Future Work .....	36
	Bibliography .....	38
	Appendix .....	40

## LIST OF FIGURES

Figure		Page
1.1	A Data Warehouse Structure .....	4
2.1	A Simplified ER Schema for a Sales Fact .....	9
2.2	A Typical Star Schema for a Sales Fact .....	9
4.1	Dataflow Topology in a Data Warehouse .....	14
4.2	Workflow of Dimension Updates .....	15
4.3	Abstract Dimension Schema & Instance .....	17
4.4	Abstract Dimension Instance Updates .....	20
4.5	Dimension Inconsistency .....	21
6.1	Creation of a Data Warehouse Dimension Table .....	32
6.2	Source Databases Installed in SQL Server 7.0 .....	32
6.3	Star Schema of a Sample Data Warehouse .....	33
6.4	Data Cubes in SQL Server 7.0 OLAP Services .....	34
6.5	Dimension Hierarchies of a Data Cube .....	34
6.6	Performance of Delta Updates Algorithm .....	35
6.7	Performance of Optimized Insertion Operation .....	35

# CHAPTER 1

## INTRODUCTION

### 1.1 Data Warehouse Concepts

Relational databases are used widely to maintain data that document everyday operations. Recently, the importance of decision support applications has increased significantly as organizations found it necessary to use the data they collected through their operational systems for future planning and decision making. Traditional databases and spreadsheets do not fully support the requirements for advanced data analysis. In the early 1990s, data warehouses emerged as systems providing information for strategic decision making. Mr. W. Inmon, the father of data warehousing, provided the formal definition in 1992: the data warehouse is an integrated, subject-oriented, time-variant, non-volatile database that provides support for decision making. The data in a data warehouse are pulled together from various data sources, and classified around meaningful entities (subjects). Data warehouses contain both historical (5 to 10 years old) and nearly current data. Once data enter the data warehouse, they are relatively static. The data warehouse is always growing [1].

Figure 1.1 shows a typical data warehouse structure [2]. A data warehouse needs to support data access (extraction process) from many sources such as database systems (e.g. a relational database), external data sources (e.g. information from other company), files of standard applications (e.g. Excel) and other document (e.g. WWW). They are usually

heterogeneous, so they need to be cleaned (finding and resolving inconsistencies) and transformed among different data formats before being loaded into the data warehouse.

Possible applications on data warehouses include [3]:

- multidimensional analysis (OLAP , On-Line Analytic Processing)
- report and query tools
- geographic information systems (GIS)
- data mining (finding knowledge from dataset with unknown patterns)
- decision support systems (DSS)
- executive information systems (EIS)
- statistics

The data warehouse provides the best opportunity for data analysis. OLAP is the vehicle to carry out the analysis. In today's data warehousing environment, OLAP becomes an integral part.

## **1.2 Data Warehouse Refreshment**

The refreshment of a data warehouse is an important process which determines the usability of the data collected and aggregated from various sources. After the initial loading process, we can maintain the data warehouse and keep it up-to-date by using two methods [4]:

- UPDATE – incremental maintenance, the process of identifying differential changes in the source tables and propagating the changed data to the data warehouse

- RELOAD – full refreshment , the process of completely erasing the contents of tables and reloading with fresh data

RELOAD is a much simpler option than UPDATE, but the jobs can take a long time to run. Compared to sizes of data warehouses (usually from several GegaBype to TeraBype), the changes are generally very small in scale. Reprocessing tens or hundreds of millions of rows to accommodate a small volume of changed data is extremely expensive. UPDATE is a cheaper option.

There are two methods to identify the changes [5]:

- Source-driven refreshment is triggered by the changed source systems.
- Client-driven refreshment is triggered on demand by users.

The refreshment processes can be carried out in the following two ways

- Immediate refresh: refreshment once the source changes are committed.
- Delayed refresh: batch refreshment at regular intervals, for example, every day, every week etc



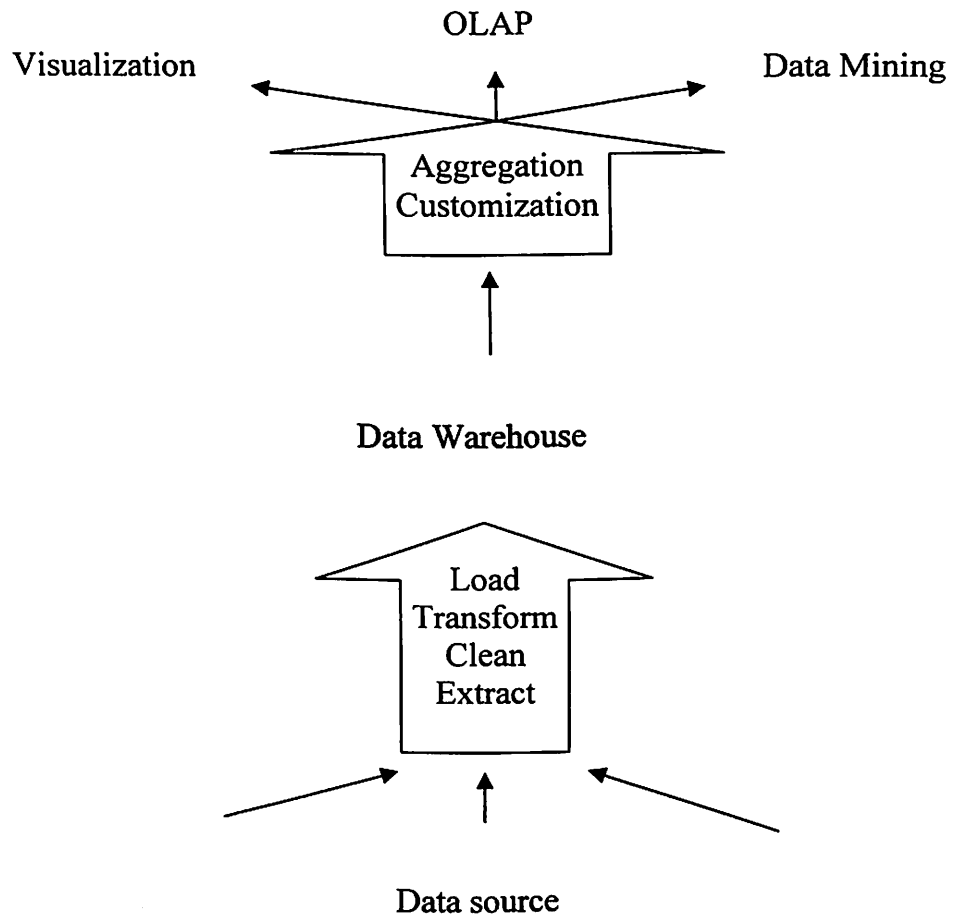


Figure 1.1: A Data Warehouse Structure

## CHAPTER 2

### DIMENSIONAL DATA MODEL

#### 2.1 Entity Relationship Model vs. Dimensional Model

Online Transaction Processing (OLTP) is profoundly different from data warehousing. [6] listed all differences from various aspects, such as users, data content, data structure and administration etc. Table 2.1 shows some of important different characteristics.

Table 2.1 Differences between Data Warehouse Data and Operational Data

Data Warehouse Data	Operational Data
Long time frame	Short time frame
Static	Rapid changes
Data is usually summarized	Record –level access
Ad hoc query access	Standard transactions
Updated periodically	Updated in real time

Entity Relationship Model (ER Model) removes data redundancy, ensures data consistency and expresses microscopic relationships. It should be used in all legacy OLTP applications based on relational technology. This is the best way to achieve the best transaction performance and the highest ongoing data integrity. Any transaction that

changes data only needs to touch the database in one place. This is the secret behind the phenomenal improvement in transaction processing speed since the early 1980s [7]. Figure 2.1 shows a simple ER model [8].

ER models designed to provide efficient data access for large numbers of transactions with very few records failed in the decision support systems which involve complex queries over very large numbers of records. There are some problems with ER model [9]:

- An ER model cannot be easily understood or remembered
- Software often cannot effectively query a general ER model
- Poor performance on retrieval of data.

A dimensional data model contains the same information as an ER model but packages the data in a symmetric format in order to achieve user friendliness, query performance and resilience to change. The main components of a dimensional data model are fact tables and dimension tables. Dimensional modeling is the only viable technique of delivering data to end users in a data warehouse [9]. Figure 2.2 shows a typical dimensional data model. There is a single fact table and four dimension tables. Each dimension table interacts with the fact table in a one-to-many relationship.

There are three physical architectures in the dimensional model according to the representations of the fact table and the dimensional tables [10]: Star Schema, Snowflake Schema and Galaxy Schema. Among them, Star Schema is the most popular one. I will introduce Star Schema in detail in the following section.

## 2.2 Star Schema

Star Schema typically has one large central table (called the fact table) and a set of smaller tables (called dimension tables) in a radial pattern around the fact table. Figure 2.2 is a simple star schema for supermarket sales data. The sales fact table is in the center. Around this fact table are four dimension tables of product, time, store and promotion.

### 2.2.1 Fact Table

A fact table stores measures and a composite key, and contains the lowest level of transaction information that is of interest to the end users.

A fact is a subject that needs to be analyzed to understand its behavior. Measures (also called fact attributes, or just facts) represent the properties of the fact that the user wants to optimize. Measures are usually numeric and additive data, and can be summarized (or aggregated) in various ways in order to extract further information. There are three types of measures according to additivity [11]:

- Full-additive: can summarize it by adding values together across any dimensions.
- Semi-additive: can be summarized across some dimension.
- Non-additive: cannot be added together across any dimensions.

A full-additive measure is desired because there are no limitations on how to use it., Semi-additive measures can be stored in the fact tables, but we must pay special attention on how to use them. A non-additive measure is unacceptable. The solution is to break it down into its full additive components, or we can use serial numbers to summarize the records.

A fact table has a composite key made up of two or more foreign keys, always indicating a many - to - many relationship.

### 2.2.2 Dimension Table

A dimension table stores hierarchical attributes, non-hierarchical attributes and a primary key.

The dimension tables are strongly denormalized and are used to select measures of interest based on user queries. The best attributes are textual, discrete, and used as the source of constraints and row headers in the user's answer set [7]. A dimension may be partitioned into a hierarchy of levels. Those attributes that organize a hierarchy of its dimension table are hierarchical attributes, which define the aggregation granularity. On the other hand, the attributes that do not organize a hierarchy are non-hierarchical attributes, which contain additional information about the hierarchical attributes. They cannot be used for aggregation. For instance, aggregating sales according to the address of the store would not make sense.

Each dimension table has a primary key that corresponds exactly to one of the component of the composite key in the fact table.

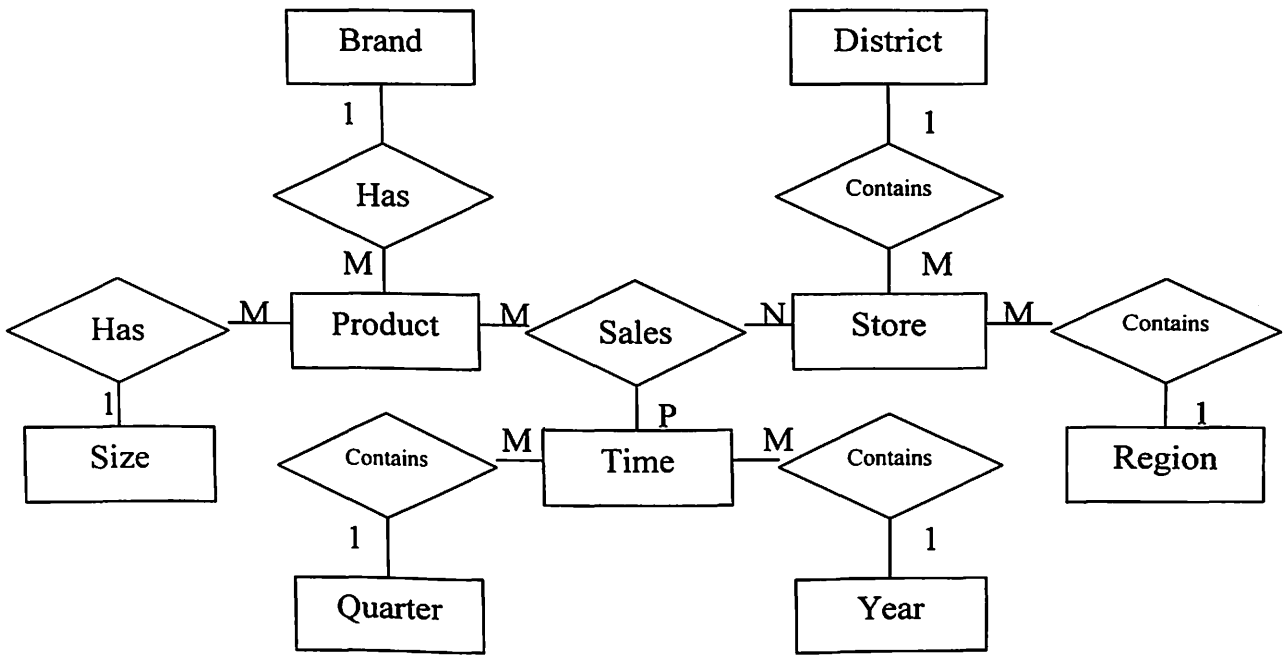


Figure 2.1: A Simplified ER Schema for a Sales Fact

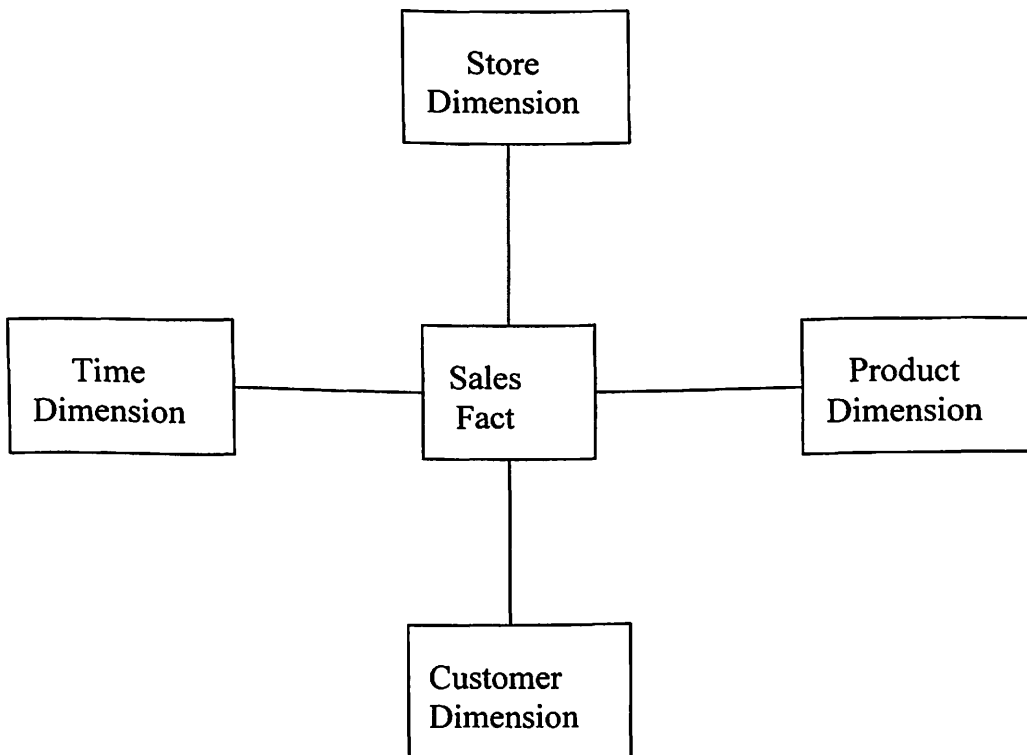


Figure 2.2: A Typical Star Schema for a Sales Fact

## **CHAPTER 3**

### **RELATED WORK**

After the initial process of loading data into a data warehouse, we can use reload or incremental update method to maintain the data warehouse. The fact table is changed frequently with the increase in the number of rows. The dimension tables are relatively static compared to the fact tables. A dimension table may change with the increase in the number of rows or with changes to the attributes. Most research on data warehouses focuses on fact tables and incremental view maintenance. However, the changes in dimension instances should also be taken into consideration.

In [7], Ralph Kimball identifies three types of slowly changing dimensions according to the maintenance technique: Type 1: overwriting the old value; Type 2: creating an additional record; Type 3: creating a new field within the old record to record the new value. Microsoft presents alternative definitions of Kimball's dimension types based on the most important and distinguishing characteristics: reflection of historical events.

- Type 1: rewriting history;
- Type2: keeping tracking of history;
- Type 3: keeping track of versions of parallel.

[12, 13] introduced a multidimensional model that includes a framework for dimension updates and a set of dimension update operators. Algorithm implementations

focus on the comparison of the normalized and the denormalized structures. However, they did not mention the base data warehouse updates.

[14] introduced the consistency of insertion and reclassification operations, defined the conflict level in each operation and explained the properties of the operation conflicts and applications in traditional and temporal data warehouses. Nonetheless, there is no formal algorithm and practical implementation associated with the insertion and reclassification operations.

[15] proposed summary delta method for incremental maintenance of aggregation views defined over the same base table. However, the work mostly focuses on the updates of the fact table. It has only a short discussion of the updates of the dimension tables.

In this thesis, I propose a dimension update model for maintaining a data warehouse, and a general dimension update strategy with the goal of reducing the system unavailable time and simplifying update processes. The remainder of this thesis is organized as follows: In chapter 4, I introduce the dimension update model, divide the update jobs into two phases Base Warehouse Updates and Cubes Updates, and formally define the dimension concepts and its properties. Chapter 4 introduces the dimension update strategy with algorithms and examples. A simulation study of the update strategy, based on an experimental implementation, is presented in Chapter 6. Conclusions and future work appear in Chapter 7.



## **CHAPTER 4**

### **DIMENSION UPDATE MODEL**

#### **4.1 Dataflow in Data Warehouse**

A simple data warehouse structure is presented in Figure 1.1. In the following paragraph, I propose a dimension update model in data warehouses. The overall dataflow is shown in Figure 4.1. A data warehouse is a combination of three different components as follows:

- Data staging area
- Data warehouse
- Data marts

The data staging area is a set of database tables that will be used to prepare the source data from the operational systems for use in a data warehouse. It provides a simple environment from which we can create data transformation and load the data into the warehouse. There is no query and presentation services in the data staging area. Both normalized and denormalized structures are acceptable.

The data warehouse is a set of data tables that will contain the same information as an OLTP system, the lowest transaction level of source data in a star schema format. We call this stage the base warehouse to reduce terminology confusion in the remaining part of the thesis.

A data mart is a logical subset of a data warehouse that has a hierarchical structure typically referred to as a cube. It can be used only in specific department and contains only the data which are relevant for the said department. A data mart enables faster response to queries. Cubes contain measures and dimensions. Measures are typically derived from the fact table, whereas dimensions are derived from the dimension tables. The advantages of building a dependent data mart [23] rather than simply letting users launch queries against a base data warehouse is as follows

- The performance is better. The limited scope of the data mart tends to make end-users navigation easier. Or a business unit may have additional requirement for data or functionality that the corporate data warehouse does not address.
- Flexibility to change or add additional data structures while minimizing the impact on end users.

There are two possible exceptions: 1) If the cost and the time of creating an enterprise base data warehouse are considered, data marts provide solutions for smaller businesses as well as focused segments of a company's decision maker. 2) Data analysis is performed directly on the base warehouse. These kinds of data flows are shown in Figure 4.1 using the dotted lines. I do not consider these exceptions in this thesis.

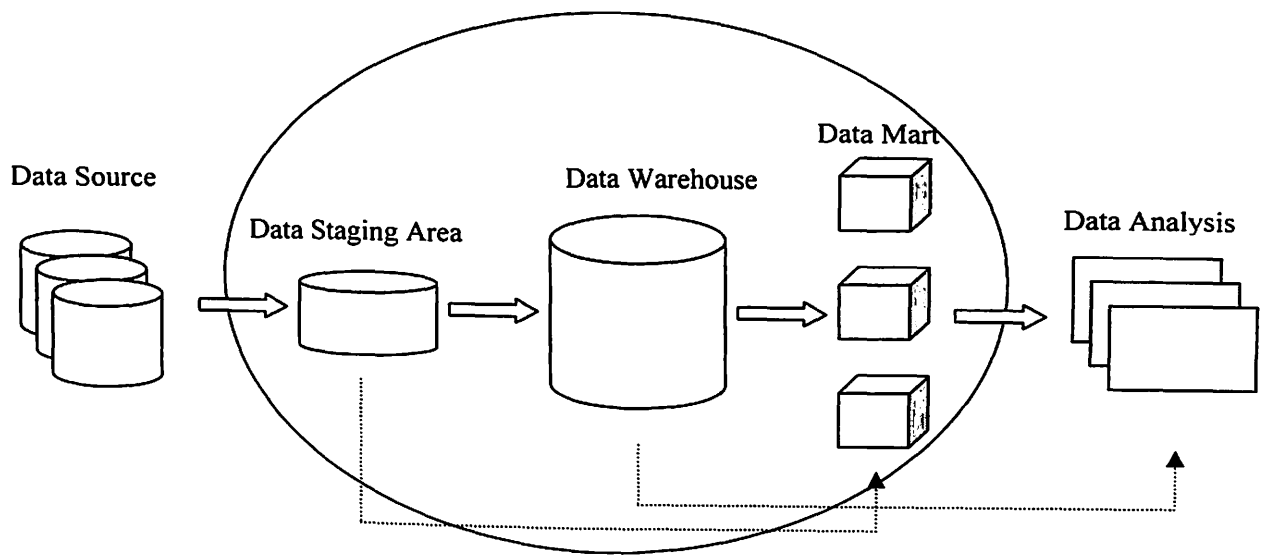


Figure 4.1 Dataflow Topology in a Data Warehouse

## 4.2 Dimension Maintenance

A data warehouse uses a star schema to contain information. The main components of a star schema are fact tables and dimension tables. A fact table is changed frequently with the increase in the number of rows. A dimension table is relatively static. In this thesis, I focus on the dimension update strategy.

According to the dataflow model we proposed, the dimension maintenance process can be divided into two phases:

- Phase I: Base dimension updates – maintaining the base data warehouse dimensions.
- Phase II: Cube dimension updates – maintaining the data mart dimensions.

Updating the base dimensions is somewhat different from updating cube dimensions. The base dimension contains the same information as the operating data in a denormalized structure, and it does not have to be a hierarchy. Cube dimension must be

defined as part of a hierarchy. A non-hierarchical dimension attribute such as color is stored in an atomic dimension table which contains only one level. In the following sections, I will give formal definitions of a cube dimension and its properties,

We use Figure 4.2 to demonstrate the process of updates. The monitors are responsible for identifying changes in the source data, and notifying the base warehouse. The integrator receives the source data, performs any necessary operations, and tells the base warehouse to store the data. After updating the base dimension, the next step is to update all cube dimensions.

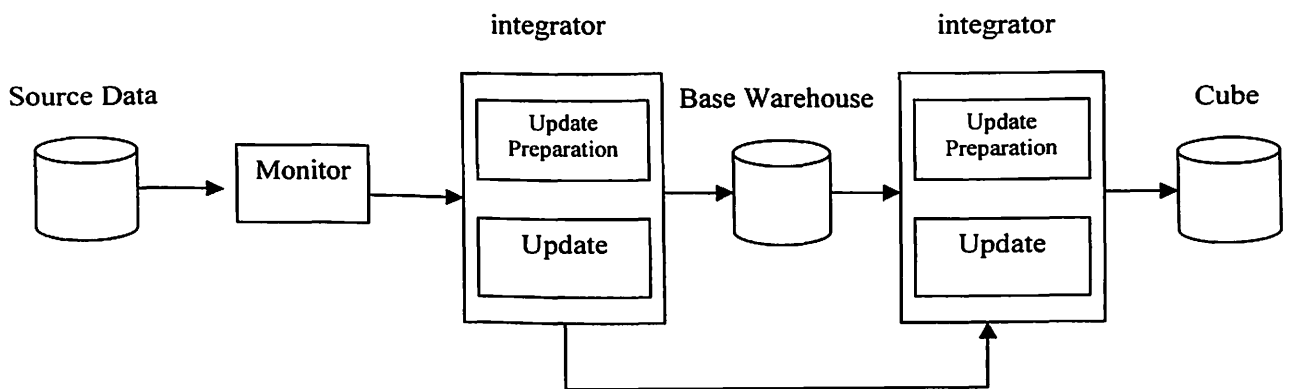


Figure 4.2: Workflow of Dimension Updates

We review the fundamental notions used in the following sections [17, 18]:

- **Functional dependency:** Let A and B be attributes in a database. B is said to be functionally dependent on A or A functionally determines B, denoted by  $A \rightarrow B$ , if and only if for each specific value  $a \in A$ , there exists exactly one value  $b \in B$ .
- **Partial order:** A relation R on a set L is reflexive, asymmetric and transitive, we call L partial order. The symbol " $\leq$ " denotes the direct relation, and " $\leq^*$ " denotes the indirect relation.

### 4.3 Dimension Schema & Instance

Dimension is a structural attribute acting as an index for identifying measures. A dimension is organized into a hierarchy of levels. The below definitions are based on the multidimensional model [12, 14, 19, 20].

#### Definition 4.1 (Dimension Schema (DS))

Dimension Schema is a directed acyclic graph  $D_S = (L_S, R_S)$ , Where:

$D_S$ : schema name

$L_S$ : a set of levels (nodes)

$R_S$ : a set of roll-up functions (directed edges)

With the following properties:

- 1) a unique bottom level  $l_t$ ,  $\text{indegree}(l_t) = 0$ ;  
a unique implicit top level  $l_{all}$ ,  $\text{outdegree}(l_{all}) = 0$
- 2)  $\exists l_i, l_j \in L, l_i \rightarrow l_j$ , then there is no other path from  $l_i$  to  $l_j$
- 3) Every level has a value domain, denoted  $\text{dom}(l)$ . For example,  $\text{dom}(l_{all}) = \{ \text{all} \}$

#### Definition 4.2 (Dimension Instance (DI))

Dimension Instance over a dimension schema  $D_S = (L_S, R_S)$  is a directed acyclic graph  $D_I = (L_I, R_I)$ , Where:

$D_I$ : instance name

$L_I$ : a set of disjoint level sets (nodes)

$R_I$ : a set of roll-up functions (directed edges)

With the following properties:

- 1)  $L_1 = M_{l_1} \cup M_{l_2} \cup \dots \cup M_{l_n}$ , where  $M_{l_i}$  is the member set of  $l_i \in L_S$
- 2)  $\exists l_i, l_j \in L, l_i \rightarrow l_j$ , then every member of  $l_i$  is connected one member of  $l_j$

Example 4.1:

Figure 4.3 (a) shows an abstract dimension schema. There are 4 levels,  $L = \{T, A, B, C\}$  the bottom level is T. The roll-up functions are  $R = \{T \rightarrow A, T \rightarrow B, A \rightarrow C, B \rightarrow C, C \rightarrow \text{all}\}$ .

Figure 4.3 (b) shows an abstract dimension instance based on Figure 4.3 (a).  $L_1 = \{\{t_1, t_2\}, \{a_1, a_2\}, \{b_1\}, \{c_1, c_2\}\}$ ,  $R_1 = \{t_1 \rightarrow a_1, t_2 \rightarrow a_2, t_1 \rightarrow b_1, t_2 \rightarrow b_1, a_1 \rightarrow c_1, a_2 \rightarrow c_2, b_1 \rightarrow c_1, b_1 \rightarrow c_2, c_1 \rightarrow \text{all}, c_2 \rightarrow \text{all}\}$

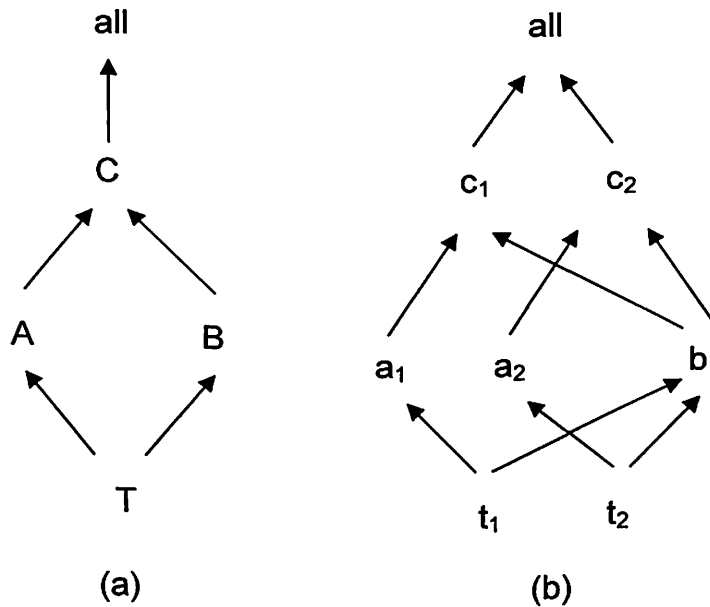


Figure 4.3: Abstract Dimension Schema & Instance

#### 4.4 Dimension Storage

We use the relational model to store dimensions. A single flat table represents a dimension instance.

Definition 4.3 (Dimension Representation (DR))

A dimension over a dimension schema  $D_S = (L_S, R_S)$  is represented by a single flat table,  $Row = (LR, RR)$ , where:

LR is a set of levels,  $LR \subset L_S$

RR is a set of relations. The relation is denoted by “->“, meaning functional dependency,  $RR \subset R_S$

Column = (LC, RC), where:

LC is a member set of a level

RC is a set of relations between members of a level. The relation is denoted by “||“ (meaning independency) or “#” (meaning dependency).

Example 4.2:

The abstract dimension instance showed in Figure 4.3 (b) can be stored in table 4.1 as follows:

Table 4.1: Dimension Representation Example

T	A	B	C
t1	a1	b1	c1
t2	a2	b1	c2

#### 4.5 Dimension Operations

In this thesis, we study dimension instance update. The terms, dimension instance update and dimension update, will be used interchangeably. Because any dimension change can be interpreted as a series of insertions and deletions, I present a formal structure of dimension updates based on the two primary operations.

The insertion operation inserts a new member into a level and connects the member to one member of every level immediately above this level. The deletion operation deletes a member of a level and moves all outgoing edges.

#### Definition 4.4 (Instance Insertion)

Given a dimension  $D_I = (L_I, R_I)$ ,  $\exists L_k \subset L_I$ , the set of level functionally determined by  $L_k$  is represented as follows:  $\text{Successor}(L_k) = \{ L_x \mid L_k \preceq L_x \}$ . After inserting a member  $l_a$  to the level  $L_k$ ,  $l_a \in L_k$ ,  $l_a \preceq l_x$  where  $l_x \in \text{Successor}(L_k)$

#### Definition 4.5 (Instance Deletion)

Given a dimension  $D_I = (L_I, R_I)$ ,  $\exists L_k \subset L_I$ ,  $l_a \in L_k$ . the set of level functionally determined by  $L_k$  is represented as follows:  $\text{Successor}(L_k) = \{ L_x \mid L_k \preceq L_x \}$ . The set of level functionally determining  $L_k$  is represented as follows:  $\text{Predecessor}(L_k) = \{ L_y \mid L_y \preceq L_k \}$ . After deleting the member  $l_a$  of the level  $L_k$ ,  $l_a \notin L_k$ ,  $l_y \preceq l_x$  where  $l_x \in \text{Successor}(L_k)$ ,  $l_y \in \text{Predecessor}(L_k)$

#### Example 4.3

Figure 4.4 (a) shows an insertion operation.  $t_3$  is inserted to level T of the dimension, adding the edges:  $t_3 \rightarrow a_1$ ,  $t_3 \rightarrow b_1$ . Figure 4.4 (b) shows a deletion operation.  $t_2$  is deleted from level T of the dimension, the edges  $t_2 \rightarrow a_2$ ,  $t_2 \rightarrow b_1$  are also been deleted.



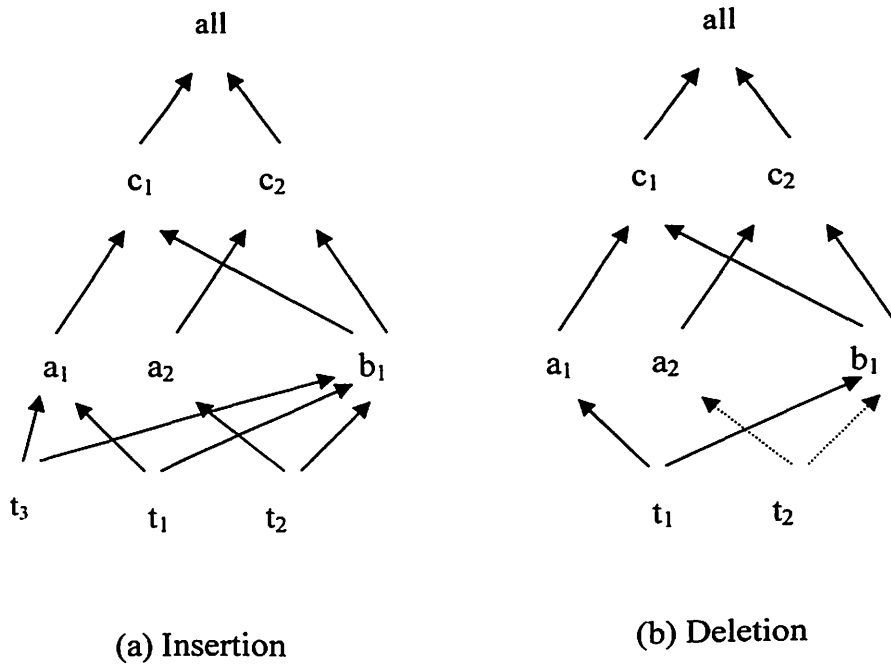


Figure 4.4: Abstract Dimension Instance Updates

#### 4.6 Dimension Consistency

In data warehouse, data consistency is a big concern. The current load should be a full and consistent set of data. When updating the dimension, we need to make sure that the dimension is consistent after any change.

Definition 4.6 (Dimension Consistency)

A dimension over a dimension schema  $D_S = (L_S, R_S)$  is consistent if  $\exists li, lj \in L, li \preceq^+ lj$ , and from one member of  $li$  it can lead to the same member of  $lj$  no matter how many paths exists between them.

Example 4.4:

Figure 4.3 (b) shows a consistent dimension instance. If  $b_1 \rightarrow c_1$  is deleted, then the dimension is not consistent because  $t_1 <^+ c_1$  and  $t_1 <^+ c_2$  at the same time. Figure 4.5 shows the result.

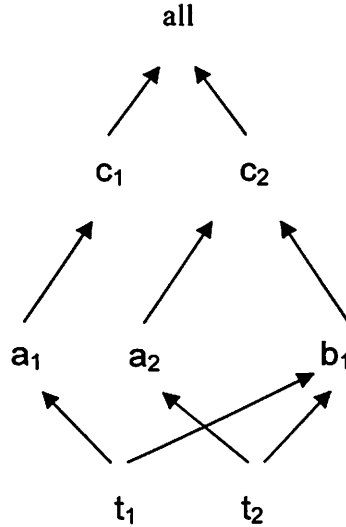


Figure 4.5: Dimension Inconsistency

## **CHAPTER 5**

### **DIMENSION UPDATE STRATEGY**

#### **5.1 Motivation**

A data warehouse can be formally understood as layers of materialized views (A view is a derived relation defined in terms of base relation. A view can be materialized by storing its extent in the database [16]). Efficient view maintenance is a big challenge. There are much research effort in this area [12, 15, 16, 21].

The data warehouse is unavailable to user queries when it is updated. The important factor as the performance measure is the system unavailable time. For efficient updates, we use “managed load updates” [9] compared to “transactional updates” of operational systems. The update process triggered by users is deferred and applied to the data warehouse in large batches at regular interval. For example, source changes received during the day are applied to the data warehouse in a nightly batch window. Nowadays the cycle of the updates is getting shorter and the time window available for making the data warehouse update is required to be shrunken more and more [21]. Moreover, many businesses have international operations in multiple time zones, so there is no convenient down time, a “night” or “weekend” when all of updates can be batched and processed together. Hence, it is critical to select an efficient update strategy.

We have made the following assumptions or restrictions in the dimension update model we proposed in previous chapter.

- The source data in the staging area have been cleaned and transformed in a normalized structure before being applied to the update process.
- There is only one base data warehouse in a star schema, which contains the lowest level data as the operating data. The dimension tables are strongly denormalized. A dimension may not be partitioned into a hierarchy of levels.
- Data cubes are subsets of the base data warehouse. A dimension is organized into a hierarchy of levels. An atomic dimension (a dimension with just one level) contains a non-hierarchical attribute.
- For purpose of simplicity, I assume that both the data warehouse and the sources use the relational model.

The goal is to minimize the time needed for updates, i.e. minimize the unavailable time of the data warehouse, and to simplify the process of updates. The general strategy is:

- Phase I – Base Dimension updates

Split the update work into two functions: propagate and refresh. The propagation function is to prepare for the changes and does not affect the base warehouse; the refresh function applies the changes to the data warehouse.

- Phase II – Cube Dimension updates

Recompute the dimension from the scratch, i.e. from the parent dimension table in the base dimension.

## **5.2 Dimension Update Algorithm**

### **5.2.1 Base Dimension Updates**

The size of a base warehouse is commonly much bigger in scale than that of its changes. Moreover, the most important difference between the OLTP data source and the data warehousing is the data structure. Recomputation from the scratch every time is too expensive. We usually incrementally update the base warehouse instead of full refreshment.

The update procedure is divided into propagate and refresh functions. The propagate function computes a delta table, which can take place without locking the data warehouse so that the data warehouse can continue to be made available for querying. The refresh function locks and updates the data warehouse from the delta table. The goal of the propagation function is to do as much work as possible so that the time required by the refresh function is minimized.

#### Definition 5.1 (Base Relation)

Let a relation  $P = \{R_1, R_2, \dots, R_n\}$ , If  $P_1$  and  $P_2$  are two relations,  $P_1$  is used to define  $P_2$ , we call  $P_1$  a Base Relation.

#### Definition 5.2 (Delta Relation)

For every relation  $P$ , relation  $\Delta P$  contains the changes made to  $P$ . we call  $\Delta P$  Delta Relation.

#### Propagate Function:

Let  $R_i$  ( $1 \leq i \leq n$ ) is a base relation;

$R_i'$  is the relation  $R_i$  to which the change of  $R_i$ ;

$\Delta R_i$  is the delta relation;

## Delta Table

$$\begin{aligned}\Delta T &= (\Delta R_1 \bowtie R_2 \bowtie \dots \bowtie R_n) U \\ &\quad (R_1' \bowtie \Delta R_2 \bowtie \dots \bowtie R_n) U \\ &\quad \dots\dots \\ &\quad (R_1' \bowtie R_2' \bowtie \dots \bowtie \Delta R_n)\end{aligned}$$

Proof by induction:

1) The basis step

Let  $n = 1$ , then the delta algorithm is correct

$$\Delta T_1 = \Delta R_1$$

Let  $n = 2$ , then the delta algorithm is correct

$$\Delta T_2 = (\Delta R_1 \bowtie R_2) U$$

$$(R_1' \bowtie \Delta R_2)$$

2) The inductive step

Suppose  $n = k$ , the delta algorithm holds

$$\Delta T_k = (\Delta R_1 \bowtie R_2 \bowtie \dots \bowtie R_k) U$$

$$(R_1' \bowtie \Delta R_2 \bowtie \dots \bowtie R_k) U$$

.....

$$(R_1' \bowtie R_2' \bowtie \dots \bowtie \Delta R_k)$$

Then for  $n > k$ ,

$$\begin{aligned}
\Delta T_{k+1} &= (\Delta R_1 \bowtie R_2 \bowtie \dots \bowtie R_k \bowtie R_{k+1}) \cup \\
&\quad (R_1' \bowtie \Delta R_2 \bowtie \dots \bowtie R_k \bowtie R_{k+1}) \cup \\
&\quad \dots \\
&\quad (R_1' \bowtie R_2' \bowtie \dots \bowtie \Delta R_k \bowtie R_{k+1}) \cup \\
&\quad (R_1' \bowtie R_2' \bowtie \dots \bowtie R_k' \bowtie \Delta R_{k+1}) \\
&= (\Delta T_k \bowtie R_{k+1}) \cup \\
&\quad (R_1' \bowtie R_2' \bowtie \dots \bowtie R_k' \bowtie \Delta R_{k+1})
\end{aligned}$$

We can conclude that the delta algorithm is correct.

Refresh Function:

Input: delta table  $\Delta T$

Output: revised dimension table D

```

For each tuple t in  $\Delta T$ 
  If ( t.flag = INSERT)
    Insert tuple t into D
  Else
    (Select *
     From D d
     Where d.id = t.id)
      if (t.flag = UPDATE)
        update tuple d using tuple t
      else if (t.flag = DELETE)
        delete tuple d from D

```

Notes about source data:

- Using date and time information to find what records have changed since the last update. If the OLTP database does not have a field that includes data/time

information, and one cannot be added, another table can be added with this information.

- Having a flag field to indicate the type of changes is very meaningful since in the data warehousing application, most changes are to insert new records. If the flag indicate INSERT operation, the new record is directly appended to the base warehouse dimension table without searching the whole table to check the existence of the record.

### 5.2.2 Cube Dimension Updates

In a data warehousing system, there may be many cubes with shared or non-shared dimensions. Updating all dimension tables is very complicated. According to our model, cube dimensions are subsets of the base warehouse dimension. No matter how big the cube dimension table is, it is simple and efficient to recompute the cube dimension table from the unique updated parent base dimension table since we need not cross-join between tables.

Update Function:

```
CREAT VIEW cube_dimension_name ( field list) AS
SELECT field lists
FROM base_dimension_name
```

For example:

A base data warehouse dimension table:

Time(time\_key, date, day\_of\_week, month , quarter, fiscal\_period, year, holiday\_flag)



A data cube dimension table:

Time( time\_key, year, quarter, month)

```
CREATE VIEW Time (time_key, year, quarter, month) AS
```

```
SELECT T.time_key, T.year, T.quarter, T.month
```

```
FROM Time T
```

## **CHAPTER 6**

### **DIMENSION UPDATE IMPLEMENTATION**

The implementation was run on an IBM computer with Intel Pentium II processor 298MHz, 128M RAM. SQL server 7.0 database management system, running on top of a Windows NT Operating System, is used as the data warehouse environment. [22, 23, 24, 25]

#### **6.1 Dimension Update Model Simulation**

We use a sample application to simulate the proposed dimension update model. The sample data were transferred from an order entry online transaction processing system (OLTP) to a base data warehouse. Then the base warehouse is used as a base relation to create data cubes for online analytical processing (OLAP).

As the intermediate database between the operating system and the data warehousing on the staging area, an OLTP database from [22] is employed, which is an order entry by multiple salespersons, simply called OLTP. Suppose OLTP has been installed as a SQL Server 7.0 database in a normalized structure. Then we use Data Transformation Services (DTS) to transform the OLTP data to the base data warehouse, called DataWarehouse. For example: we create the dimension table “store” from two tables “store” and “region” of OLTP database using SQL statement. Figure 6.1 shows the transformation. Similarly,

we can create other dimension tables. Two SQL Server 7.0 databases OLTP and DataWarehouse were shown in Figure 6.2.

The DataWarehouse database included one fact table called “sales\_fact” and five dimension tables, called “customer”, “product”, “time”, “store”, and “promotion”. The star schema for the base databases is presented in Figure 6.3.

Based on the DataWarehouse database, we created three cubes using SQL Server OLAP Services, called “Department1”, “Department2”, “Department3”, shown in Figure 6.4. The cubes are created according to the user’s requirements. For example: The “Department1” users want to analyze the consumption behavior of customers. There are two dimension tables, one is “customer” and the other is “time”, shown in Figure 6.5. Both dimensions have the characteristics and properties introduced in the previous chapter.

## **6.2 Experiments and Performance**

Experiments were carried out to test the performance improvement obtained by optimized update method. First, I used delta algorithms to get the delta table from the changed test database, and record the propagation time; then I applied the net changed data represented in the delta table to the dimension tables in the data warehouse, and record the refresh time. The implementation was done using Visual Basic.NET, the completely object-oriented programming environment.

The test database is the same as the one used in the example described in section 6.1. Suppose there are three operations on the test database: insertion, deletion and update. Since most changes are to insert new records to the data warehouse in common business

practices, I arbitrarily assume that the ratio of operation number is 94% insertions, 4% updates and 4% deletions of total changed records respectively. For example, the delta table has 100 rows. Then 94 rows will be inserted to the dimension table, 4 rows will be deleted from the dimension table and 4 rows will be updated in the dimension table,

I assume each tuple in the delta table causes a single update to the dimension table and each tuple in the dimension table is updated at most once.

Figure 6.7 plots the variation in elapsed time as the size of the set changes for a fixed size of the data warehouse. I varied the size of the changes from 100 tuples to 800 tuples. We can see that the system downtime of both methods increases with the increase in the size of the changes. The graph illustrates the performance advantage of using the delta updates method.

I optimized the refresh function by adding a flag field in the source table to indicate the new rows. When the cursor in the delta table finds one record is new, it can be directly appended to the dimension table in the data warehouse, without searching the whole dimension table to check if there exists a matching tuple in the dimension table. The performance improvement is shown in Figure 6.8. I varied the size of insertion changes from 100 tuples to 800 tuples. We can see that the system downtime increases with the increase in the size of insertion changes. The optimized method outperformed the original method. Moreover, the improved margin appears to be higher when the size of the insertion changes increases.

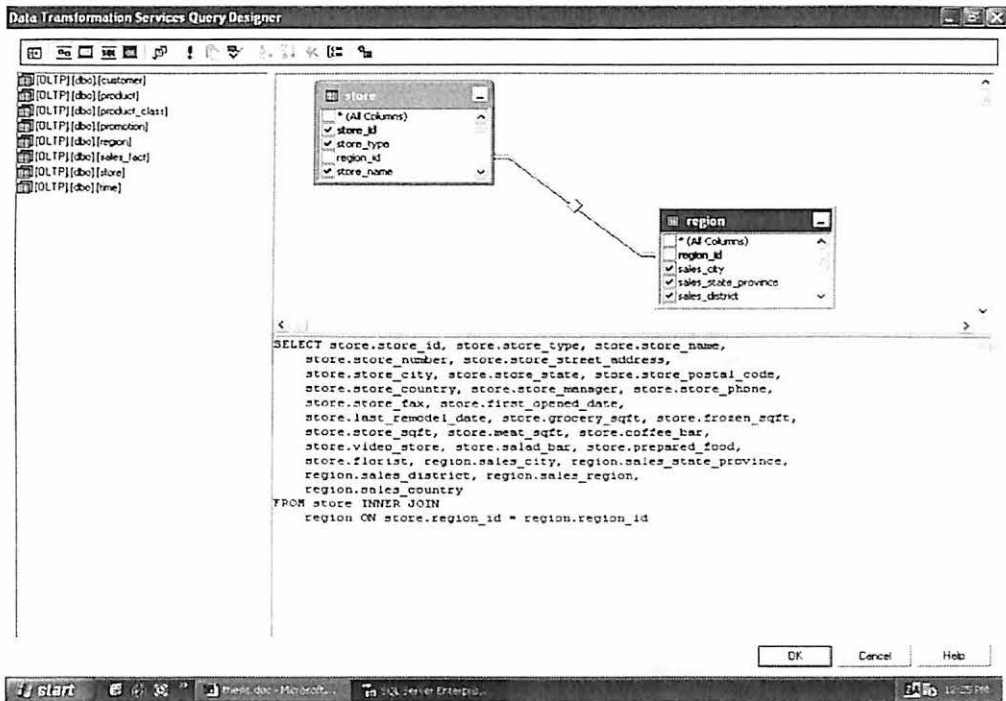


Figure 6.1 Creation of a Data Warehouse Dimension Table

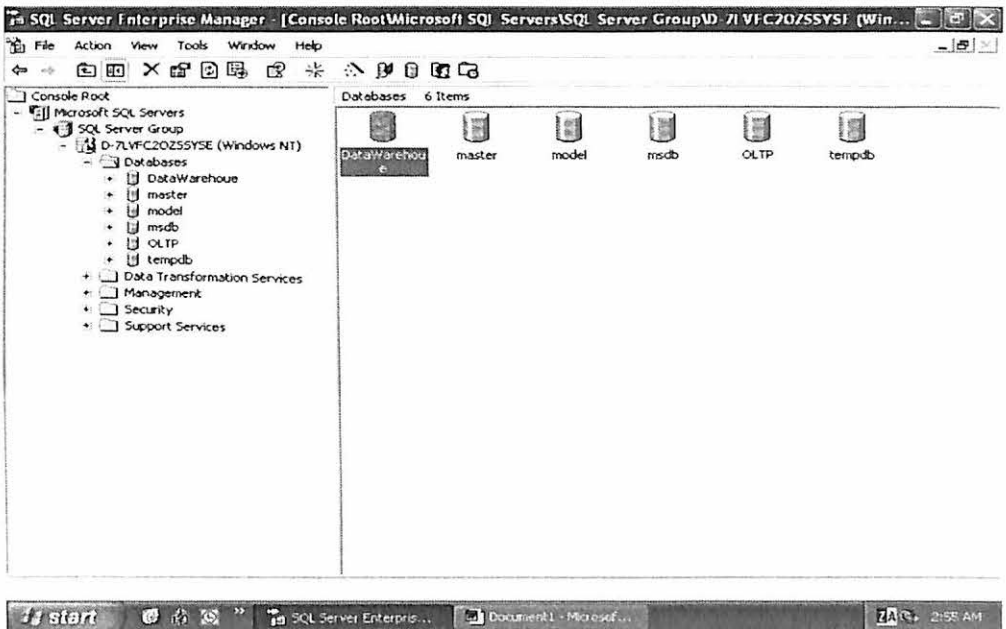


Figure 6.2 Source Databases Installed in SQL Server 7.0

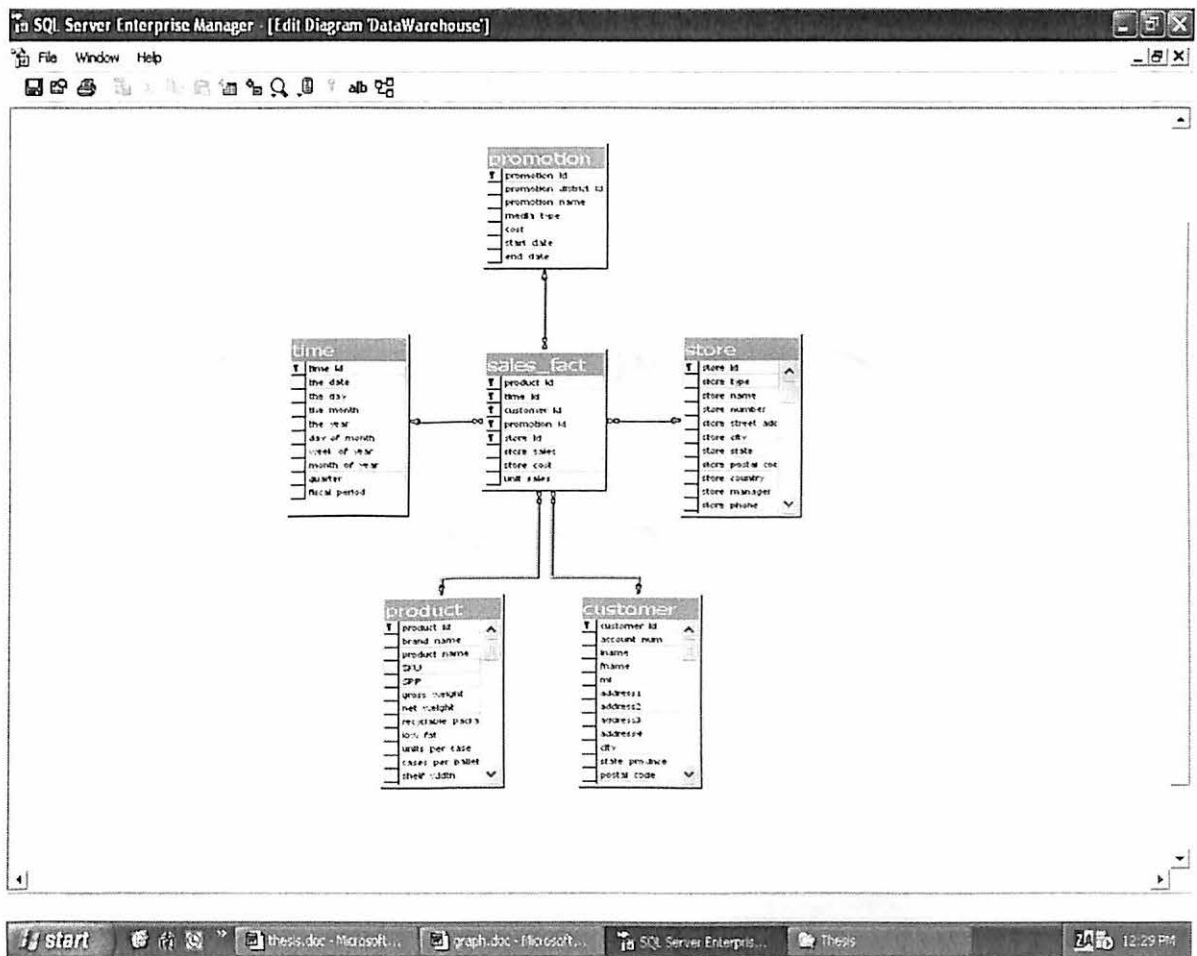


Figure 6.3 Star Schema of a Sample Data Warehouse

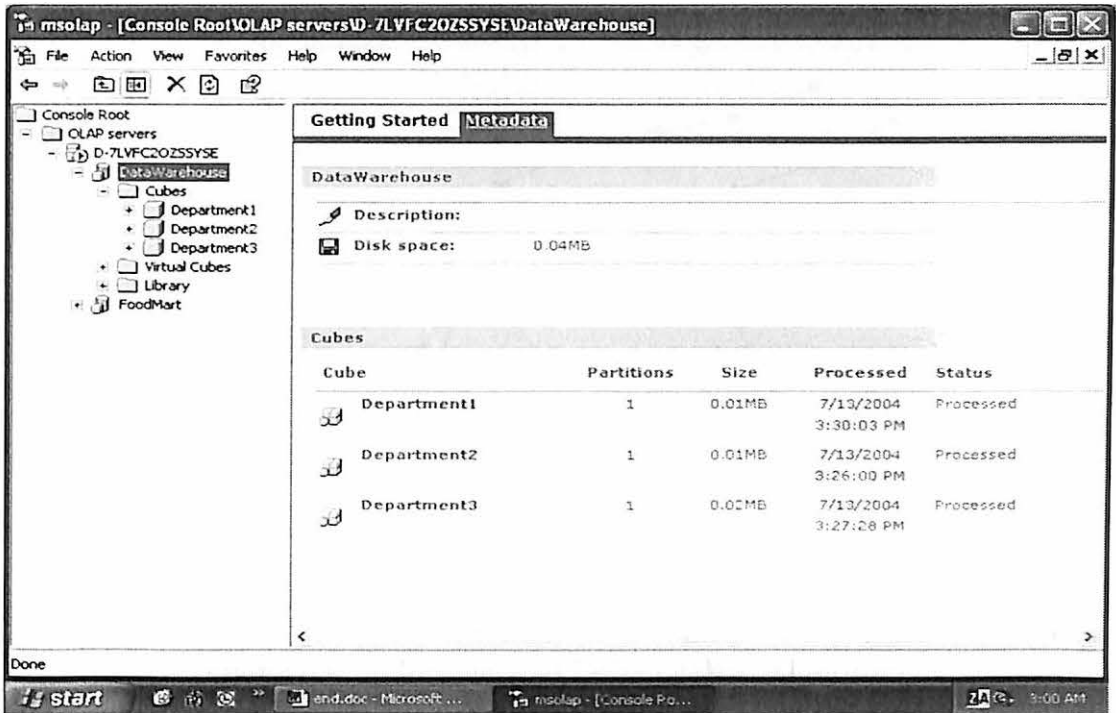


Figure 6.4 Data Cubes in SQL Server 7.0 OLAP Services

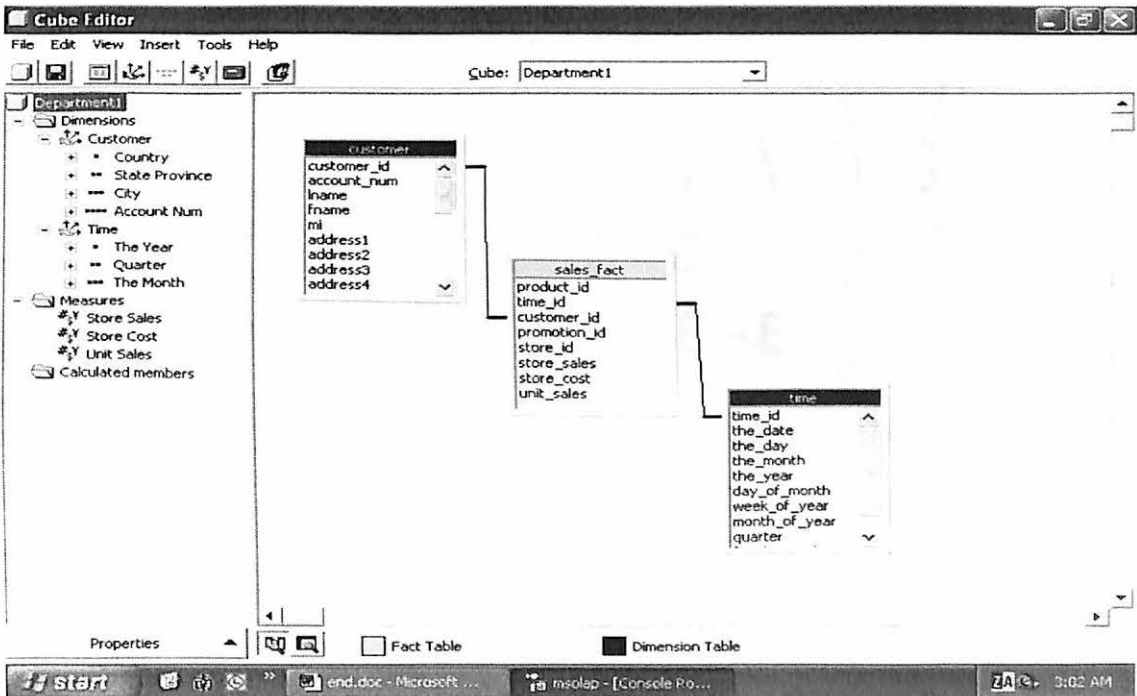


Figure 6.5 Dimension Hierarchies of a Data Cube

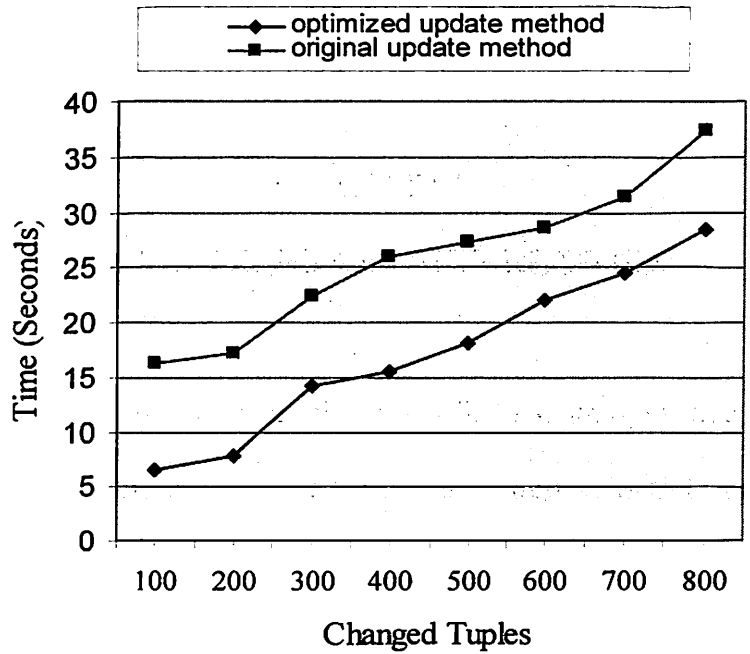


Figure 6.6 Performance of Delta Update Algorithm

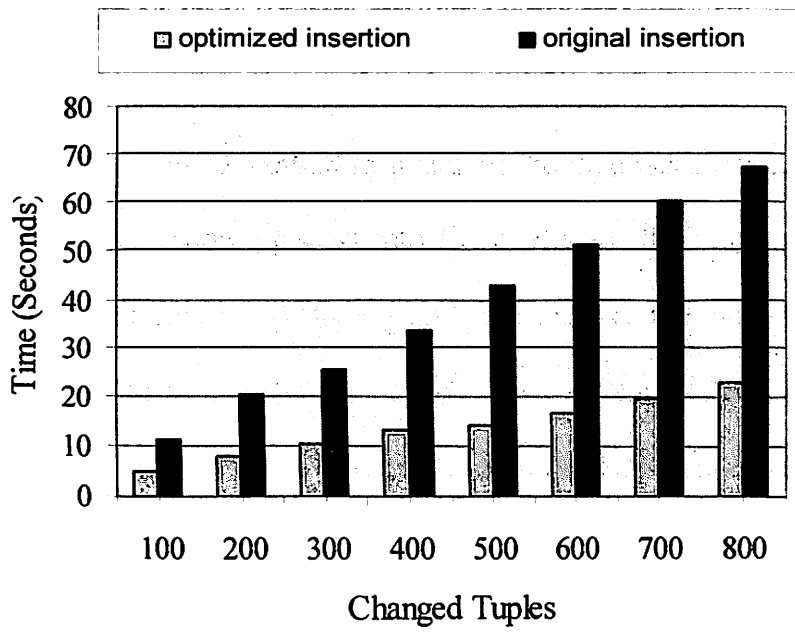


Figure 6.7 Performance of Optimized Insertion Operation



## CHAPTER 7

### CONCLUSIONS AND FUTURE WORK

A data warehouse is a user-centered environment for data analysis and decision support. Dimensional modeling is the only viable technique to deliver data to end users in a data warehouse. Star schema is the most popular form of dimensional modeling. The main components of a star schema are fact tables and dimension tables. The fact table is changed frequently with the increase in the number of rows. The dimension tables are relatively static compared to the fact tables. Nonetheless, the changes should also be taken into considerations.

After the initial process of loading data into a data warehouse, we can use the fully refresh or the incremental update method to maintain the data warehouse. The dimension update model divides the update process into three phases: the staging area, the data warehouse and the data marts. In the staging area, we prepare the data of changes using the delta algorithm so as to reduce the data warehouse downtime. The data warehouse is an intermediate repository for efficiently updating all cubes. The data cube is a subset of the data warehouse. We take advantage of the relationship between the base warehouse and the data mart to update cube dimensions by using the simple yet efficient full refresh method without imposing a significant overhead.

We divide dimension update procedure into propagate and refresh functions. The propagate function computes a delta table; an activity that can take place without locking

the data warehouse so that the data warehouse can continue to be made available for querying. The experiment results show that the delta updates algorithm is efficient to reduce the data warehouse downtime. The experiments also show that the optimization for insertion operation in the refresh function is significant.

The following issues will be focused on in the future work:

- Type II of slowly changing dimensions [7,9]: keep a record of the old dimension data. For example, when an employee is moved from branch to branch, all of the old transactions should be in the old branch, and only the new transactions should be related to the new branch.
- In general, there are many materialized views based on one or some data cubes, or even the base data warehouse. If we can reuse the results of the delta tables to update the materialized views, we can considerably improve the overall performance.

## BIBLIOGRAPHY

- [1] Jeffrey A. Hoffer, Mary B. Prescott and Fred R. McFadden, *Modern Database Management*, 6<sup>th</sup> edition, Prentice Hall 2002.
- [2] Harry Singh. *Data Warehousing Concepts, Technologies, Implementations, and Management*. Prentice-Hall 1998.
- [3] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou and Panos Vassiliadis *Fundamentals of Data Warehouses*, Springer 1998.
- [4] Paulraj Ponniah. *Data Warehousing Fundamentals*. John Wiley & Sons 2001.
- [5] Donald J. Berndt and John W. Fisher, *Understanding Dimension Volatility in Data Warehouses*, the Sixth INFORMS Conference on Information Systems and Technology (CIST-2001). November 3-8, 2001.
- [6] Thomas M. Connolly and Carolyn E. Begg, *Database Systems: A Practical Approach to Design, Implementation, and Management*, second edition, Addison-Wesley 1998.
- [7] Ralph Kimball, *The Data Warehouse Toolkit*. John Wiley & Sons 1996.
- [8] Michael Drippendorf and Il-Yeol Song, *The Translation of Star Schema into Entity-Relationship Diagrams*. Proceedings of the 8<sup>th</sup> International Workshop on Database and Expert Systems Applications. September 1 -2, 1997. Pages 390-395.
- [9] Ralph Kimball, Laura Reeves, Margy Ross and Warren Thornthwaite. *The Data Warehouse Lifecycle Toolkit*. John Wiley & Sons 1998.
- [10] Sung Ho Ha and Sang Chan Park, *Data Modeling for Improving Performance of Data Mart*, IEMC'98. Pages 436 – 441.
- [11] Matteo Golfarelli, Dario Maio and Stefano Rizzi. *Conceptual Design of Data Warehouses from E/R Schemes*. Proceedings of the 31<sup>st</sup> Annual Hawaii International conference on System Sciences – Volume 7, 1998. Pages 334.
- [12] Carlos A. Hertado, Alberto O. Mendelzon and Alejandro A.Vaisman. *Maintaining Data Cubes under Dimension Updates*. In Proceedings of the IEEE-ICDE Conference 1999. Pages 346-355.
- [13] Carlos A. Hertado, Alberto O. Mendelzon and Alejandro A.Vaisman. *Updating OLAP Dimensions*. Proceedings of the 2<sup>nd</sup> ACM International Workshop on Data Warehousing and OLAP (DOLAP) 1999. Pages 60 – 66.

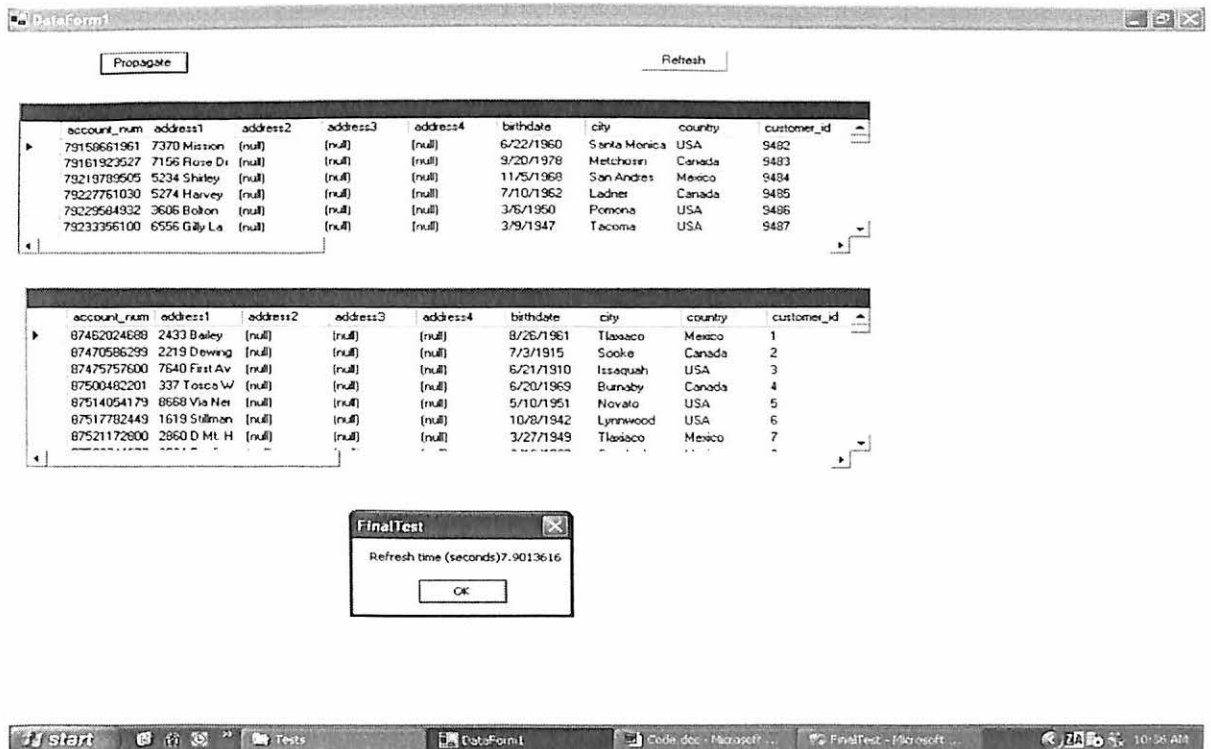
- [14] Carolin Letz, Eric Tobias Henn and Gottfried Vossen. *Consistency in Data Warehouse Dimensions*. International Database Engineering and Applications Symposium. July 17 -19, 2002. Pages 224 – 232.
- [15] Inderpal S. Mumick, Dallan Quass and Barinderpal S. Mumick. *Maintenance of Data Cubes and Summary Tables in a Warehouse*. ACM SIGMOD Volume 26, Issue 2. June, 1997. Pages 100-111.
- [16] Ashish Gupta, Inderpal S. Mumick and V.S. Subrahmanian. *Maintaining Views Incrementally*. Proceeding of ACM SIGMOD Conference on Management of Data, 1993. Pages 157-166.
- [17] W. Lehner, J. Albrecht and H. Wedekend. *Normal Forms for Multidimensional Databases*. Proceedings of the 10<sup>th</sup> international Conference on Scientific and Statistical Database Management. Pages 63 – 72.
- [18] Raghu Ramakrishnan. *Database Management Systems*. McGraw-Hill 1998.
- [19] Luca Cabibbo and Riccardo Torlone. *Querying Multidimensional Databases*. Proceedings of the 6<sup>th</sup> International Workshop on Database Programming Languages (DBPL'97), 1997. Pages 319 – 335.
- [20] Luca Cabibbo and Riccardo Torlone. *A Logical Approach to Multidimensional Databases*. Proceedings of the 6<sup>th</sup> International Conference on Extending Database Technology: Advances in Database Technology. Pages 183 – 187.
- [21] Ki Yong Lee, Jin Hyun Son and Myoung Ho Kim. Efficient Incremental View Maintenance in Data Warehouses. CIKM'01, November 5-10, 2001. Pages 349 - 356.
- [22] Jake Sturm. *Data Warehousing with SQL Server 7.0: Technical Reference*. Microsoft Press 2000.
- [23] Robert S. Craig, Joseph A. Vivona and David Bercovitch. *Microsoft Data Warehousing: Building Distributed Decision Support Systems*. John Wiley & Sons, Inc. 1999.
- [24] Michael Corey, Ian Abramson, Larry Barnes, Benjamin Taub and Rajan Venkitachalam. *SQL Server 7 Data Warehousing*. Osborne/McGraw-Hill 1999.
- [25] Sakhr Youness, *Professional Data Warehousing with SQL Server 7.0 and OLAP Services*. Wrox Press Ltd. 2000.

## APPENDIX A

### Source Codes

The program is to record the system elapsed time for dimension update process. It is implemented in Visual Basic .NET

The source tables are changed OLTP tables. The target tables are dimension tables which need to be updated. The program carried out two major tasks: propagate and refresh functions. The sample window is as follows. After pushing the “Refresh” button, we can get the time of updating the target table. Similarly, we can get the propagate time.



```

Import system.data
Import system.data.oledb
Import system.windows.forms

```

```

'The public class DataForm1 is the major body of the program
'In this class, main method is called to facilitate the program run.
'All the object are defined and instantiated in this class.
'Database access operations are set up.
'ADO.NET is employed to get the data connections, data adapters,
'datasets, up and running.
'For each data command object, some parameters are established to
'fill the "?" in the SQL commands embedded.

```

```

Public Class DataForm1
    Inherits System.Windows.Forms.Form

```

```

    Public Sub New()
        MyBase.New()
        InitializeComponent()
    End Sub

```

```

'Form overrides dispose to clean up the component list.

```

```

Protected Overrides Sub Dispose(ByVal disposing as_
    Boolean)
    If Not (components Is Nothing) Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub

```

```

'declare object variables

```

```

Private components As System.ComponentModel.IContainer
Friend WithEvents OleDbSelectCommand1 As OleDbCommand
Friend WithEvents OleDbConnection1 As OleDbConnection
Friend WithEvents OleDbDataAdapter1 As OleDbDataAdapter
'FinalTest.customer1 is pre-defined dataset
Friend WithEvents objcustomer1 As FinalTest.customer1
Friend WithEvents btnLoad As Button
Friend WithEvents btnUpdate As Button
Friend WithEvents grdcustomer As DataGridView
Friend WithEvents OleDbDataAdapter2 As OleDbDataAdapter
Friend WithEvents OleDbSelectCommand2 As OleDbCommand
Friend WithEvents OleDbInsertCommand2 As OleDbCommand
Friend WithEvents OleDbUpdateCommand2 As OleDbCommand
Friend WithEvents OleDbDeleteCommand2 As OleDbCommand
Friend WithEvents OleDbConnection2 As OleDbConnection
'FinalTest.customer2 is pre-defined dataset
Friend WithEvents Customer21 As FinalTest.customer2
Friend WithEvents DataGridView1 As DataGridView

```

```

'initialize objects

<System.Diagnostics.DebuggerStepThrough(> Private sub_
InitializeComponent()
    Me.OleDbSelectCommand1 = New OleDbCommand
    Me.OleDbConnection1 = New OleDbConnection
    Me.OleDbDataAdapter1 = New OleDbDataAdapter
    Me.objcustomer1 = New FinalTest.customer1
    Me.btnLoad = New Button
    Me.btnUpdate = New Button
    Me.grdcustomer = New DataGridView
    Me.OleDbDataAdapter2 = New OleDbDataAdapter
    Me.OleDbSelectCommand2 = New OleDbCommand
    Me.OleDbInsertCommand2 = New OleDbCommand
    Me.OleDbUpdateCommand2 = New OleDbCommand
    Me.OleDbDeleteCommand2 = New OleDbCommand
    Me.OleDbConnection2 = New OleDbConnection
    Me.Customer21 = New FinalTest.customer2
    Me.DataGrid1 = New DataGridView
    CType(Me.objcustomer1, _
        System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.grdcustomer, _
        System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.Customer21, _
        System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.DataGrid1, _
        System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()

    Me.OleDbSelectCommand1.CommandText = "SELECT * FROM customer_
        where flag=1 or flag=2 or flag=3"
    Me.OleDbSelectCommand1.Connection = Me.OleDbConnection1

    Me.OleDbConnection1.ConnectionString = "Data
        Source=""localhost\OLTP1.mdb"";"

    Me.OleDbDataAdapter1.SelectCommand = Me.OleDbSelectCommand1
    Me.OleDbDataAdapter1.TableMappings.AddRange(New _
        System.Data.Common.DataTableMapping() {'itemize data table and
        dataset relationship field by field})

    Me.objcustomer1.DataSetName = "customer1"
    Me.objcustomer1.Locale = New
        System.Globalization.CultureInfo("en-US")

    Me.btnLoad.Location = New System.Drawing.Point(80, 24)
    Me.btnLoad.Name = "btnLoad"
    Me.btnLoad.TabIndex = 0
    Me.btnLoad.Text = "&Propagate"

    Me.btnUpdate.Location = New System.Drawing.Point(544, 24)
    Me.btnUpdate.Name = "btnUpdate"
    Me.btnUpdate.TabIndex = 1
    Me.btnUpdate.Text = "&Refresh"

    Me.grdcustomer.DataMember = "customer"
    Me.grdcustomer.DataSource = Me.objcustomer1

```

```

Me.grdcustomer.HeaderForeColor = _
    System.Drawing.SystemColors.ControlText
Me.grdcustomer.Location = New System.Drawing.Point(10, 76)
Me.grdcustomer.Name = "grdcustomer"
Me.grdcustomer.Size = New System.Drawing.Size(734, 164)
Me.grdcustomer.TabIndex = 3

Me.OleDbDataAdapter2.DeleteCommand = Me.OleDbDeleteCommand2
Me.OleDbDataAdapter2.InsertCommand = Me.OleDbInsertCommand2
Me.OleDbDataAdapter2.SelectCommand = Me.OleDbSelectCommand2
Me.OleDbDataAdapter2.TableMappings.AddRange(New_
    System.Data.Common.DataTableMapping() {'itemize data table and
        dataset reflection relationship field by field})
Me.OleDbDataAdapter2.UpdateCommand = Me.OleDbUpdateCommand2

Me.OleDbSelectCommand2.CommandText = "SELECT * FROM customer"
Me.OleDbSelectCommand2.Connection = Me.OleDbConnection2

Me.OleDbInsertCommand2.CommandText = "INSERT INTO
customer('list all the field names) VALUES ('list "?" )"
Me.OleDbInsertCommand2.Connection = Me.OleDbConnection2
Me.OleDbInsertCommand2.Parameters.Add(New
System.Data.OleDb.OleDbParameter("account_num",
System.Data.OleDb.OleDbType.Double, 0, "account_num"))
'Add parameters like the above statement for every field.
'Here not every field is listed for brevity purpose since they
'are pretty similar and can be easily derived

Me.OleDbUpdateCommand2.CommandText = "UPDATE customer SET
account_num = ?, 'add assignment like the preceding one for every
field WHERE (customer_id = ?) AND (account_num = ? OR ? IS NULL
AND account_num IS NULL) AND 'conditional statements like
preceding ones for each field"
Me.OleDbUpdateCommand2.Connection = Me.OleDbConnection2
Me.OleDbUpdateCommand2.Parameters.Add(New_
System.Data.OleDb.OleDbParameter("account_num",_
System.Data.OleDb.OleDbType.Double, 0, "account_num"))
'Add parameters like the above statement for every field.
'Here not every field is listed for brevity purpose since they
'are pretty similar and can be easily derived

Me.OleDbDeleteCommand2.CommandText = "DELETE FROM customer WHERE
(customer_id = ?) AND (account_num = ? OR ? IS NULL AND
account_num IS NULL) 'conditional statements like preceding
ones for each field "
Me.OleDbDeleteCommand2.Connection = Me.OleDbConnection2
Me.OleDbDeleteCommand2.Parameters.Add(New_
System.Data.OleDb.OleDbParameter("Original_
customer_id",
System.Data.OleDb.OleDbType.Integer, 0,
System.Data.ParameterDirection.Input, False, CType(0, Byte),
CType(0, Byte), "customer_id",
System.Data.DataRowVersion.Original, Nothing))
'Add parameters like the above statement for every field.
'Here not every field is listed for brevity purpose since they
'are pretty similar and can be easily derived

```



```

Me.OleDbConnection2.ConnectionString = "Data
Source=""localhost\OLTP2.mdb"";"

Me.Customer21.DataSetName = "customer2"
Me.Customer21.Locale = New System.Globalization.CultureInfo("en-
US")

Me.DataGrid1.DataMember = ""
Me.DataGrid1.DataSource = Me.Customer21
Me.DataGrid1.HeaderForeColor =
System.Drawing.SystemColors.ControlText
Me.DataGrid1.Location = New System.Drawing.Point(16, 272)
Me.DataGrid1.Name = "DataGrid1"
Me.DataGrid1.Size = New System.Drawing.Size(728, 184)
Me.DataGrid1.TabIndex = 4

Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(776, 478)
Me.Controls.Add(Me.DataGrid1)
Me.Controls.Add(Me.btnLoad)
Me.Controls.Add(Me.btnUpdate)
Me.Controls.Add(Me.grdcustomer)
Me.Name = "DataForm1"
Me.Text = "DataForm1"
CType(Me.objcustomer1, System.ComponentModel.ISupportInitialize).
_EndInit()
CType(Me.grdcustomer, System.ComponentModel.ISupportInitialize).En
dInit()
CType(Me.Customer21, System.ComponentModel.ISupportInitialize).End
Init()
CType(Me.DataGrid1, System.ComponentModel.ISupportInitialize).EndI
nit()
Me.ResumeLayout(False)

```

End Sub

```

'This subroutine handles the button click event.
'In this case, when "Refresh" button is clicked, this will be invoked.
'It calls dataset update procedure and calculate time required for the
'refresh process. Finally, it displays time using message box.

```

```

Private Sub btnUpdate_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles btnUpdate.Click
    Dim cc As DateTime
    Dim dd As DateTime
    Dim ff As decimal
    Try
        cc = DateTime.Now
        Me.UpdateDataSet()
        dd = DateTime.Now
        ff = (dd.Ticks - cc.Ticks)/10000000.00
        MsgBox(cc.ToString())
        MsgBox(dd.ToString())
        MsgBox("Refresh time (seconds) "&ff.ToString())
    Catch eUpdate As System.Exception
        System.Windows.Forms.MessageBox.Show(eUpdate.Message)
    End Try

```

End Sub

'This subroutine handles the button click event.  
'When "Propagate" button is clicked, this will be invoked.  
'It calls data loading procedure and calculate time required for the  
'propagate process. Finally, it displays time using message box.

```
Private Sub btnLoad_Click(ByVal sender As System.Object, ByVal e As_  
System.EventArgs) Handles btnLoad.Click
```

```
    Dim aa As DateTime
```

```
    Dim bb As DateTime
```

```
    Dim ee As Decimal
```

```
    Try
```

```
        'Attempt to load the dataset.
```

```
        aa = DateTime.Now
```

```
        Me.LoadDataSet()
```

```
        bb = DateTime.Now
```

```
        ee = (bb.Ticks - aa.Ticks)/1000000.0
```

```
        MsgBox(aa.ToString())
```

```
        MsgBox(bb.ToString())
```

```
        MsgBox("Propagate time (seconds) "&ee.ToString())
```

```
    Catch eLoad As System.Exception
```

```
        System.Windows.Forms.MessageBox.Show(eLoad.Message)
```

```
    End Try
```

End Sub

' This procedure implement operations on the dataset of target table.  
' Several data tables are created in order to carry out the task  
' Data rows are used and their collection objects are also employed  
' The index is the flag field of the source table. Using the index,  
' the procedure finds relevant records in target to work on or, if  
' the record is new, it appends the record behind the target dataset.

```
Public Sub UpdateDataSet()
```

```
    Dim i As Integer
```

```
    Dim j As Integer
```

```
    Dim y As Integer = 0
```

```
    Dim a As DataTable = New System.Data.DataTable
```

```
    Dim b As DataTable = New System.Data.DataTable
```

```
    Dim c As DataTable = New System.Data.DataTable
```

```
    Dim m As DataRow
```

```
    Dim n(100) As DataRow
```

```
    a = objcustomer1.Tables.Item("customer")
```

```
    b = Customer21.Tables.Item("customer")
```

```
    For i = 0 To a.Rows.Count - 1
```

```
        m = a.Rows.Item(i)
```

```
        If m("flag") = 1 Then
```

```
            Dim x As DataRow = b.NewRow()
```

```
            x("customer_id") = m("customer_id")
```

```
            x("account_num") = m("account_num")
```

```
            x("lname") = m("lname")
```

```
            x("fname") = m("fname")
```

```
            x("mi") = m("mi")
```

```
            x("address1") = m("address1")
```

```
            x("address2") = m("address2")
```

```
            x("address3") = m("address3")
```

```
            x("address4") = m("address4")
```

```

x("city") = m("city")
x("state_province") = m("state_province")
x("postal_code") = m("postal_code")
x("country") = m("country")
x("customer_region_id") = m("customer_region_id")
x("phone1") = m("phone1")
x("phone2") = m("phone2")
x("birthdate") = m("birthdate")
x("marital_status") = m("marital_status")
x("yearly_income") = m("yearly_income")
x("gender") = m("gender")
x("total_children") = m("total_children")
x("num_children_at_home") = m("num_children_at_home")
x("education") = m("education")
x("date_acct_opened") = m("date_acct_opened")
b.Rows.Add(x)
ElseIf m("flag") = 2 Then
    Dim r As DataRow
    For Each r In b.Rows
        'For j = 0 To b.Rows.Count - 1
        If r("customer_id") = m("customer_id") Then
            y = y + 1
            n(y - 1) = r
            Exit For
        End IF
    Next r
ElseIf m("flag") = 3 Then
    For j = 0 To b.Rows.Count - 1
        If b.Rows.Item(j)("customer_id") = m("customer_id") Then
            b.Rows.Item(j)("account_num") = m("account_num")
            b.Rows.Item(j)("lname") = m("lname")
            b.Rows.Item(j)("fname") = m("fname")
            b.Rows.Item(j)("mi") = m("mi")
            b.Rows.Item(j)("address1") = m("address1")
            b.Rows.Item(j)("address2") = m("address2")
            b.Rows.Item(j)("address3") = m("address3")
            b.Rows.Item(j)("address4") = m("address4")
            b.Rows.Item(j)("city") = m("city")
            b.Rows.Item(j)("state_province") = _
            m("state_province")
            b.Rows.Item(j)("postal_code") = m("postal_code")
            b.Rows.Item(j)("country") = m("country")
            b.Rows.Item(j)("customer_region_id") = _
            m("customer_region_id")
            b.Rows.Item(j)("phone1") = m("phone1")
            b.Rows.Item(j)("phone2") = m("phone2")
            b.Rows.Item(j)("birthdate") = m("birthdate")
            b.Rows.Item(j)("marital_status") = _
            m("marital_status")
            b.Rows.Item(j)("yearly_income") = m("yearly_income")
            b.Rows.Item(j)("gender") = m("gender")
            b.Rows.Item(j)("total_children") = _
            m("total_children")
            b.Rows.Item(j)("num_children_at_home") = _
            m("num_children_at_home")
            b.Rows.Item(j)("education") = m("education")
        End If
    Next j

```

```

                b.Rows.Item(j) ("date_accnt_opened") = _
                m("date_accnt_opened")
            Exit For
        End If
    Next
End If
Next

```

```

    Dim k As Integer
    For k = 0 To y - 1
        n(k).Delete()
    Next k
    Customer21.GetChanges()
    DataGrid1.SetDataBinding(Customer21, "customer")
    Me.UpdateDataSource(Customer21)
    Customer21.AcceptChanges()
End Sub

```

'This subroutine is used to load the source table into  
'a dataset and populate the dataset with the query results  
'called in the first data adapter.

```

Public Sub LoadDataSet()
    Dim objDataSetTemp As FinalTest.customer1
    Dim objDataSetTemp2 As FinalTest.customer2
    objDataSetTemp = New FinalTest.customer1
    objDataSetTemp2 = New FinalTest.customer2
    Try
        Me.FillDataSet(objDataSetTemp)
        Me.FillDataSet2(objDataSetTemp2)
    Catch eFillDataSet As System.Exception
        Throw eFillDataSet
    End Try
    Try
        objcustomer1.Clear()
        Customer21.Clear()
        objcustomer1.Merge(objDataSetTemp)
        Customer21.Merge(objDataSetTemp2)

        Catch eLoadMerge As System.Exception
            Throw eLoadMerge
    End Try
End Sub

```

' This subroutine update the target data table using  
' the dataset produced after dataset update procedure.  
' This one involves the interaction between the memory-residing  
' data set and the physical-drive-residing target data table.

```

Public Sub UpdateDataSource(ByVal NewObject As FinalTest.customer2)
    Try
        Me.OleDbConnection2.Open()
        OleDbDataAdapter2.Update(NewObject, "customer")
    Catch updateException As System.Exception
        Throw updateException
    Finally

```

```

        Me.OleDbConnection2.Close()
    End Try
End Sub

'This is intermediate process happened during the process of source
'table populating data set.

Public Sub FillDataSet(ByVal dataSet As FinalTest.customer1)
    dataSet.EnforceConstraints = False
    Try
        Me.OleDbConnection1.Open()
        Me.OleDbDataAdapter1.Fill(dataSet)
    Catch fillException As System.Exception
        Throw fillException
    Finally
        dataSet.EnforceConstraints = True
        Me.OleDbConnection1.Close()
    End Try
End Sub

'The below process happens when the program populate
'the data set relating to the target table.

Public Sub FillDataSet2(ByVal dataSet As FinalTest.customer2)
    dataSet.EnforceConstraints = False
    Try
        Me.OleDbConnection2.Open()
        Me.OleDbDataAdapter2.Fill(dataSet)
    Catch fillException As System.Exception
        Throw fillException
    Finally
        dataSet.EnforceConstraints = True
        Me.OleDbConnection2.Close()
    End Try
End Sub

End Class

```

VITA



Haihong Ma

Candidate for the Degree of

Master of Science

**Thesis:** DIMENSION UPDATES IN DATA WAREHOUSES

**Major Field:** Computer Science

**Biographical:**

**Personal Data:** Born in Shandong, China

**Education:** Received Bachelor of Science degree in Chemical Engineering from Dalian University of Technology, Dalian, China in July 1992. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December, 2004.

**Experience:** Jinan Oil Refinery Company, as an engineer, 1992 – 1994. Hewlett-Packard Medical Products (Qingdao) Co. Ltd, as an office clerk, 1995 - 1999.