# AN OBJECT-ORIENTED VERSIONING APPROACH FOR MULTIDIMENSIONAL DATABASE SCHEMA EVOLUTION

SOMCHART FUGKEAW

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE
(COMPUTER SCIENCE)
FACULTY OF GRADUATE STUDIES
MAHIDOL UNIVERSITY
2004

ISBN 974-04-5098-9

Thesis
entitled

# AN OBJECT-ORIENTED VERSIONING APPROACH FOR MULTIDIMENSIONAL DATABASE SCHEMA EVOLUTION

……………………………………..
Mr. Somchart Fugkeaw
Candidate

………………………………………..….
Assoc.Prof. Jarernsri L. Mitrpanont, Ph.D.
Major-Advisor

………………………………………..….
Lect. Charnyote Pluempitiwiriyawej, Ph.D.
Co-Advisor

………………………………………..….
Assoc.Prof. Rassmidara Hoonsawat, Ph.D.
Dean
Faculty of Graduate Studies

………………………………………..….
Assoc.Prof. Supachai Tangwongsan, Ph.D.
Chair
Master of Science Programme in
Computer Science
Faculty of Science

Thesis
entitled

# AN OBJECT-ORIENTED VERSIONING APPROACH FOR MULTIDIMENSIONAL DATABASE SCHEMA EVOLUTION

was submitted to the Faculty of Graduate Studies, Mahidol University
for the degree of Master of Science (Computer Science)

on
June 3, 2004

# ACKNOWLEDGEMENTS

There are many people who have concerned with the achievement of this thesis. My deepest gratitude first goes to my advisor, Associate Professor Dr. Jarernsri L. Mitrpanont for her precious advice, cheerfulness, constructive guidance and encouragement. With her kind supervision of all levels of this research work, I was able to complete this thesis.

I am grateful to Dr. Panos Vasilliadis for his help and suggestions about the research issue as well as the discussion on research vision, which inspire the production of the state of the research.

I would also like to thank the members of my thesis examining committee: Dr.Charnyote Pluempitiwiriyawej, Assistant Professor Dr. Sukanya Pongsuphap, and Associate Professor Dr. Chinda Achariyakul for their review, invaluable advice and constructive feedback, which are important to my thesis.

My appreciation goes to Dr.Sudsanguan Ngamsuriyaroj for her constructive comments and kind suggestions. She gave me immense feedback on my proposed ideas and draft concepts.

It would be pleasure for me to express thank to Dr.Markus Blaschka for his dissertation: FIESTA, from which his research idea partially contributes to my research vision.

Furthermore, I would like to  thank all my colleagues and graduate friends for their invaluable encouragement, friendship and assistance during this study. And also thanks all the faculty members in the Department of Computer Science, Mahidol University, for knowledge, comments and support.

Finally, I would like to express a special thank to my parents for their financial support, unconditional love, encouragement and moral support given to me all the time. For their contribution to this thesis, I dedicate the thesis to my father, my mother and all the teachers.

Somchart Fugkeaw

AN OBJECT-ORIENTED VERSIONING APPROACH FOR MULTIDIMENSIONAL DATABASE SCHEMA EVOLUTION

SOMCHART FUGKEAW 4437174 SCCS/M

M.Sc. (COMPUTER SCIENCE)

THESIS ADVISORS: JARERNSRI L. MITRPANONT, Ph.D., CHARNYOTE PLUEMPITIWIRIYAWEJ, Ph.D.

**ABSTRACT**

This thesis proposes an alternative solution to support the schema evolution in multidimensional databases. The research solution is elaborated from three important concepts. First, we applied the object-oriented data model to represent the major structure of the multidimensional database (MDB), which yields a new multidimensional modeling scheme called Object-oriented Multidimensional Databases (OOMDB). Second, the prominent Object-oriented versioning method was applied to handle the changes of MDB schema. Finally, we adopted the MDB schema evolution operations defined in FIESTA approach as the change initiator to the underlying database schema. Then, two schema versioning evolution algorithms, namely, *Forward Schema Version Compatibility Algorithm* and *Backward Schema Version Compatibility Algorithm* were developed to conduct a reliable process for MDB schema changes and schema version retrieval, respectively. These proposed algorithms satisfy full scale support of both forward and backward schema evolution compatibility, which are the crucial properties of the versioning technique.

Also, we developed a prototype system called OOMDB SEMAN (Object-Oriented Multidimensional DataBase Schema Evolution MANager) to serve as an easy-to-use tool for MDB schema update. The tool enables a more flexible modification of MDB schema. In the experiment, Star Tracker$^{TM}$ software, which is a grocery system of real-world business, was used as the base MDB schema and the data. Additionally, we developed an OLAP (Online Analytical Processing) data generator to generate a high volume of transactional data for the test. Up to 1,000,000 records of data were tested with the 100 schema version cases. To verify our proposed solution, the Analysis Manager, which is a system module of Microsoft SQL Server 2000, was used to validate the correctness and consistency of the functionality against the OOMDB SEMAN. The result shows that all schema versions are perfectly consistent and functionally correct for schema transformation and instance propagation. Ultimately, the thesis concludes that our proposed approach offers another reasonable way for multidimensional database schema evolution support.

KEY WORDS: MULTIDIMENSIONAL SCHEMA / OBJECT-ORIENTED MULTIDIMENSIONAL MODELING / SCHEMA EVOLUTION / VERSIONING TECHNIQUE / OLAP

123 pp. ISBN 974-04-5098-9

การใช้หลักการเวอร์ชันนิ่งเชิงวัตถุ สำหรับการปรับเปลี่ยนเค้าโครงฐานข้อมูลแบบหลายมิติ

(AN OBJECT-ORIENTED VERSIONING APPROACH FOR MULTIDIMENSIONAL DATABASE SCHEMA EVOLUTION)

สมชาติ ฟักเขียว : 4437174 SCCS/M

วท.ม. (วิทยาการคอมพิวเตอร์)

คณะกรรมการควบคุมวิทยานิพนธ์:เจริญศรี มิตรภานนท์, Ph.D., ชาญยศ ปลื้มปิติวิริยะเวช, Ph.D.

## บทคัดย่อ

งานวิจัยนี้ได้นำเสนอแนวทางในการสนับสนุนและจัดการการเปลี่ยนแปลงของระบบฐานข้อมูลแบบหลายมิติ โดยแนวคิดของงานวิจัยเกิดขึ้นจากการประยุกต์ใช้หลักการที่สำคัญ 3 หลักการ ได้แก่ 1. การใช้หลักการออกแบบโมเดลเชิงวัตถุ สำหรับสร้าง เค้าโครงฐานข้อมูลแบบหลายมิติ ซึ่งก่อให้เกิดโมเดลรูปแบบใหม่ เรียกว่า เค้าโครงฐานข้อมูลแบบหลายมิติเชิงวัตถุ 2.การประยุกต์ใช้หลักการเวอร์ชั่นนิ่งเชิงวัตถุในการจัดการและควบคุมการเปลี่ยนของเค้าโครงฐานข้อมูลแบบหลายมิติ 3. การใช้และการกำหนดคุณสมบัติของการเปลี่ยนแปลงเค้าโครงฐานข้อมูลแบบหลายมิติ โดยอาศัยแนวคิดของ FIESTA ในส่วนของตัวปฏิบัติการที่ก่อให้เกิดการเปลี่ยนแปลงของเค้าโครงฐานข้อมูล และเราได้มีการนำเสนอ สองอัลกอริทึม สำหรับการทำเวอร์ชั่นนิ่ง ได้แก่ 1. Forward Schema Version Compatibility Algorithm และ 2. Backward Schema Version Compatibility Algorithm เพื่อควบคุมกระบวนการเปลี่ยนแปลงของเค้าโครงฐานข้อมูลและการเรียกคืนเวอร์ชั่นของเค้าของโครงฐานข้อมูลตามลำดับ ดังนั้น อัลกอริทึม ที่เรานำเสนอนี้สามารถสนับสนุนการเปลี่ยนแปลงของเค้าโครงฐานข้อมูลได้ทั้งก่อนและหลัง ซึ่งเป็นคุณสมบัติที่สำคัญของหลักการเวอร์ชั่นนิ่ง

นอกจากนี้แล้ว ผู้วิจัยได้สร้างระบบจำลอง OOMDB SEMAN เพื่อใช้เป็นเครื่องมือที่มีความสะดวกในการใช้งาน ในแง่ของการรองรับและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นกับเค้าโครงฐานข้อมูลแบบหลายมิติ ในการทดลองเราได้ใช้ โครงสร้างฐานข้อมูลของระบบขายของชำที่จำลองมาจากระบบจริงของซอฟต์แวร์ Star Tracker$^{TM}$ และนอกจากนี้เราสร้างระบบที่ใช้ในการสร้างรายการข้อมูลสำหรับทดสอบกับข้อมูลปริมาณมาก โดยมีข้อมูลประมาณหนึ่งล้านเรคคอร์ดสำหรับการทดสอบร่วมกับหนึ่งร้อยเวอร์ชั่น ในการตรวจสอบถึงแนวความคิดของงานวิจัยชิ้นนี้ ผู้วิจัยได้ใช้ซอฟต์แวร์ Microsoft SQL Server 2000 ในส่วนของ Analysis Manager เป็นตัวเปรียบเทียบกับระบบจำลองที่สร้างขึ้น โดยการทดสอบจะเปรียบเทียบถึง ความถูกต้อง และความสม่ำเสมอ ของหน้าที่การทำงานของระบบจำลอง โดยพิจารณาจากผลการประมวลข้อคำถามที่ส่งเข้าไปในระบบ ซึ่งผลลัพธ์แสดงให้เห็นว่าระบบจำลอง OOMDB SEMAN สามารถทำงานได้อย่างถูกต้อง และยังเป็นการแสดงให้เห็นว่าแนวความคิดของงานวิจัยชิ้นนี้มีประโยชน์ในทางปฏิบัติในการใช้ควบคุมและสนับสนุนการเปลี่ยนแปลงของเค้าโครงฐานข้อมูลแบบหลายมิติได้

123 หน้า ISBN 974-04-5098-9

# CONTENTS

# CONTENTS (CONT.)

## CONTENTS (CONT.)

# LIST OF TABLES

# LIST OF TABLES (CONT.)

# LIST OF FIGURES

Page

## LIST OF FIGURES (CONT.)

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

Recently, data warehousing technology has played a significant role in decision support scenarios. The manager performs an analytical query from the data warehouse using Online Analytical Processing (OLAP) tool. The data warehouse is typically modeled in a multidimensional view upon the dimensional axis and factual data. Although the usual assumption is that dimensions are relatively static, in practice, the dimension updates are often necessary for adapting the multidimensional database in response to the changes of requirements. These updates take place either at the structural level or at the instance level, requiring adjustments in the database schema. As a result, the schema evolution issue in data warehousing area is active and challenging.

Most research in this area concentrates on view maintenance. These approaches demonstrate how to maintain the materialized views when the data at any source has been changed. In data warehousing, view maintenance brings up many problems such as self-maintenance, consistency maintenance, update filtering and on-line view maintenance [20]. There are a number of papers dealing with these matters [15,23,30,32]. To the best of our knowledge, there are very few works dedicated to a complete methodology in designing and maintaining the structural schema.

In our research, we propose a new framework and methodology as an alternative approach of the Multidimensional Database Schema Evolution. The novel idea is rooted from three important concepts: (1) we use the Object-oriented technique to model the major structures of the MDB, rendering a new MD modeling scheme called Object-Oriented Multidimensional Databases (OOMDB); (2) the concept of Object-oriented versioning is applied to handling the changes of multidimensional database schema; and (3) we adopt the MDB schema evolution operations defined in FIESTA approach [8] as the change initiator to the underlying database schema. In fact, the OO model enables the design for MDB to be more flexible than the traditional relational model. In OO context, the MDB schema is composed of dimension class and fact class, which are easy to be analyzed and maintained. Furthermore, the schema versioning approach plays a major mechanism to support the change of OOMDB schema in a very appropriate manner since any evolved schema and its instances are well kept and can actively respond to the queries specified by the previous schema versions, if required.

As the state of the research, we introduce the two prominent algorithms, namely, *Forward Schema Version Compatibility Algorithm* and *Backward Schema Version Compatibility Algorithm*, to support MDB schema evolution and retrieval process. The former is used to support the structural changes of the MDB schema as well as the propagation of the instances. The latter is used to satisfy the backward compatibility property, which is an important feature of the versioning basis. It is exploited to control the schema version retrieval process and construct the MDB schema and its instance in a legal state of the particular schema version. In addition, all change events performed by these two algorithms are well retained for further audit. With the proposed algorithms, the update of MDB schema is perfectly supported and the structural MDB schema and its instance are guaranteed to be consistent accordingly.

We also develop a prototype system called OOMDB SEMAN (Object-Oriented Multidimensional Database Schema Evolution MANager) to verify our proposed solution. In the experiment, a real-world grocery system is used as an experimental system. The Analysis Manager, which is a system module of Microsoft SQL Server 2000, is used to validate the correctness and consistency of the functionality. The OOMDB SEMAN consists of OLAP analytical features and the administrative feature of schema evolution control. For the test scenario, the MDB schema of the Analysis Manager is constructed in respect to the structural schema of the specified version of the prototype in case-by-case fashion. And the set of analytical queries is applied to both systems. The result shows that the OOMDB SEMAN is feasible and beneficial to implement, and demonstrates that our proposed approach offers another reasonable way for multidimensional database schema evolution support.

## 1.2 An Organization of the Thesis

This thesis is organized in seven chapters as follows:

Chapter 1: Introduction

This chapter introduces the overview of this thesis.

Chapter 2: Literature Review

This chapter provides the necessary background concepts of schema evolution in object-oriented database, schema modification approaches for data warehousing and multidimensional data modeling. In addition, some works related to our research are discussed.

Chapter 3: Problem Statement

This chapter describes the problems of the existing approaches as well as the motivation of this research. It also includes the objectives and contributions of the work.

Chapter 4: An Object-Oriented Versioning Approach for Multidimensional Database Schema Evolution

This chapter formally introduces our proposed multidimensional database schema evolution framework. It describes all components constituting the framework. Then, the concepts are scrutinized in details leading to our research solution.

Chapter 5: Design and Implementation

This chapter explains the design and implementation scheme for the OOMDB SEMAN prototype system.

Chapter 6: Experiment and Result

This chapter presents the details of the experimental scenario and its result.

Chapter 7: Discussion and Conclusion

This chapter discusses and concludes the overall research solution. Also, the interesting future works are recommended.

# CHAPTER 2

# LITERATURE REVIEW

The objective of this chapter is to review the relevant research backgrounds to our proposed framework. We begin with the schema evolution which is a core subject of our research. We introduce the concept of schema evolution in object-oriented database, specifically the schema versioning approach. Then some literatures of the schema evolution in multidimensional database systems are stated. In addition, the multidimensional data models are discussed. Finally, the related approaches to our research are described.

## 2.1 Schema Evolution

### 2.1.1 Schema Evolution in Object-Oriented Databases

Object-oriented database systems provide a rich semantic data model, compared to conventional relational database systems. As explored, the schema evolution was commonly implemented with object-oriented data model. There are two approaches in schema evolution: *non-version* and *version* approaches [10]. The non-version approach is considered as the modification approach, which only regards on how the change of current state of the database schema and its instance is properly performed to the next state. This approach supports and controls the changes of database schema in one-way perspective. Although many research works have addressed non-versioned approach, Killian [28] suggested that

*"Schema evolution is versioning of types. It is very difficult and very expensive to support schema evolution without versioning"*

In versioning approach, it demonstrates a way of handling the versioning history of a schema by specifying versions and relationships between versions [10]. Versioning base is implemented in two levels: *class versioning and schema versioning*. For class versioning, the evolution handling is treated in the class level. Basically, once the change has occurred in any classes, the affected classes need to be versioned. However, the schema versioning preserves the overall schema as a version when the change occurs. In most cases, the schema versioning is preferred to class versioning because it regards the whole schema as a complete version of the database. Rather, the maintenances of version for both schema and instance are more flexible. However, more storage requirement is a drawback of schema versioning approach compared to class versioning or non-versioning approach.

In [22], schema versioning is defined as follows: *"Schema Versioning is accommodated when a database system allows the accessing of all data, both retrospectively and prospectively, through user definable version interfaces."*

For the schema evolution in OODB area; the issues on the modification of the database schema, the migration of existing instances, and ensuring a well-defined consistency criteria (such as a correct association of instances to their classes) are concerned.

### 2.1.2 Schema Modification Approaches for Data Warehousing and OLAP

One challenge in data warehousing area is the evolution issue. Most research work in this area concentrates on view maintenance. These works emphasize how to maintain the materialized views when the data at any source changes. The process of reflecting the views up-to-date in response to the changed source data is called *View Maintenance*. The view maintenance yields related problems such as self maintenance, consistency maintenance, update filtering and on-line view maintenance [20]. Therefore, many papers have addressed these matters.

Since several approaches of materialized view in data warehousing are not relevant to this thesis's objectives, we thus omit them. For further details, readers who are interested in this issue can explore in [22,23].

The followings are a few works on frameworks and methodologies to address the schema evolution issue. These approaches often focus on multidimensional database systems. Thus, literatures discussed here are selected from the relevant subject of this research. We briefly introduce the three following approaches:

- Dimension Updates of Hurtado, Mendelzon and Vaisman
- Temporal Data Warehouse Approach of Eder and Koncilia
- Multidimensional Evolution Framework: FIESTA of Marcus

#### 2.1.2.1 Dimension Updates of Hurtado,Mendelzon and Vaisman

Hurtado et al. [15, 16] proposed a formal model of dimension updates in a multidimensional data model (implemented as a materialized view). The multidimensional data model of Cabbibo and Torlone [2] is used.

They introduced a set of primitive operators to perform the update either to the schema of dimension or to dimension instances. In addition, they defined additional special operators for dimension update into two parts: the changes of the classification hierarchy and changes of the classification nodes (dimension members). Also, the study of the evolution effect over the class of materialized view and dimension levels is presented. Furthermore, algorithms are elaborated to efficiently maintain the materialized view.

#### 2.1.2.2 Temporal Data Warehouse Approach of Eder and Koncilia

The approach of Eder and Koncilia [17] aims at representing changes in dimension data of multidimensional data warehouses, by introducing temporal extension, structural versioning and transformation functions.

Basic idea is to assign a valid time interval $[T_s, T_e]$ representing the valid time beginning at $T_s$ and ending at $T_e$ (with $T_e > T_s$) to all dimension members and all hierarchical links between the dimensions.

The authors introduce a formal model of a temporal multidimensional system consisting of:

i) A number of dimensions N+1

ii) A set of dimensions $\mathbf{D} = \{D_1,\ldots, D_N, F\}$ where F is the dimension describing the required facts and $D_i$ are all other dimensions including a time dimension if required.

iii) A number of dimension members **M**.

iv) A set of dimension members $DM = DM_{D1} \cup \ldots \cup DM_{DN} \cup DM_F = \{DM_1,\ldots, DM_M\}$ where $DM_F$ is the set of all facts, $DM_{Di}$ is the set of all dimension members which belong to dimension $D_i$.

A set of hierarchical assignments $H = \{H_1,\ldots, H_O\}$ where $H_i =< DM^{C}_{id}, DM^{P}_{id}$, Level, $[T_s, T_e] >$. $DM^{P}_{id}$ is the dimension member identifier of the parent of $DM^{C}_{id}$ or 0 if the dimension member is a top-level dimension member. *Level* is a value $0\ldots L$ where *L* is the number of layers and *Level* is the level of $DM^{C}_{id}$. All "leaves" are at level 0.

Furthermore, the model of structure version is defined for recording all changes in the schema. Each structure version is a 4-tuple $<Sv_{id}, T, \{DM_{D1, SVid}, \ldots, DM_{DN, SVid}, DM_{F, SVid}\}, H_{SVid} >$ where SVid is a unique identifier and T represents the valid time of that structure version as a time interval [Ts, Te]. Conceptually, each structure version SV has a corresponding cube with the same valid time interval used for answering the given time point.

To answer the queries on the data warehouse, the user always has to define which structure version should be used. The data is returned by tracing to the specified version. Therefore, the transformation functions for mapping data from one structure version to a different structure version are also provided in this research.

### 2.1.2.3 Multidimensional Evolution Framework: FIESTA

The overall objective of FIESTA [8] is to introduce a framework that supports schema evolution for OLAP systems that are specified and managed on a conceptual level in a tool-supported environment.

The FIESTA framework focuses on the evolution of multidimensional schema by using the ME/R model as a graphical conceptual model to represent multidimensional semantic. Thus, all modifications occur on the conceptual level which is specified by a group of evolution operators. For the schema change and effect, the authors present descriptive algebra to support all phases of the design and maintenance cycle.

Since FIESTA is implemented via a graphical tool-supported environment, the author proposes a formal dualism that allows using both the algebraic and graphical representations of a given multidimensional schema equivalently. Also, the normal form for ME/R graphs and formal mappings between both representations are formally presented.

For a given schema evolution, the logical schema is adapted accordingly to conform the change of the conceptual level. By this, they provide a formal mapping between the conceptual layer and the logical layer by means of schema evolution algebra that serves as a complete formal specification for an implementation of the FIESTA framework. Additionally, they propose a transformation algorithm that transforms a sequence of conceptual schema evolution operations to a sequence of corresponding logical evolution operations.

## 2.2 Multidimensional Data Models

There are many research addressed on multidimensional data models [1,2,3,4,24,25]. However, there is no agreement for a formal MD model. Specifically, most existing approaches are based on the two underlying MD data structure, i.e., relational oriented (ROLAP) and cube oriented (MOLAP). Several OLAP products are therefore available for serving the specific multidimensional database scheme. In research trend of multidimensional modeling, several research works have tried to propose the candidate framework as a solution of MD modeling. An object-oriented model is labeled as another powerful solution for multidimensional design. In recent years, several papers [4, 7, 19] have proposed the multidimensional data model in terms of an object-oriented perspective.

Next, we briefly introduce the most prominent models grouped by three kinds of multidimensional data models:

Models based on Relational-Oriented

- Model of Li and Wang
- Model of Gyssens and Lakshmanan

Models based on Cube-Oriented

- Model of Agrawal, Gupta and Sarawagi
- Model of Cabbibo and Torlone
- Model of Vassiliadis

Models based on Object-Oriented

- Model of Trujillo and Palomar
- Model of Nguyen, Tjoa and Wagner

### 2.2.1 Models Based on Relational-Oriented

#### 2.2.1.1 Model of Li and Wang

The multidimensional model proposed by Li and Wang [24] is based on relational model. Basic concept is a multidimensional cube consisting of a number of relations called "dimension relations". The cube is constructed from the Cartesian product of the dimensions to the measurement and these results are mapped to "grouping relations". This paper introduces MD cube algebra for manipulating such cubes.

An n-dimensional cube scheme is a set $\{(D_1,R_1),\ldots,(D_n,R_n)\}$where $D_i$ is dimension name and $R_i$ is a set of attribute names. An MD cube on such a scheme is a pair (F, $\mu$) where F =$\{(D_1,r_1),\ldots,(D_n,r_n)\}$with $r_i$ being a relation or $R_i$ for each i and $\mu$ being a mapping from $\{\{(D_1,t_1,\ldots,(D_n,t_n)\}| \forall\ 1\leq i\leq n: t_i \in r_i$ $\}$ to V (set of scalar values).

A multidimensional database consists of a finite set of multidimensional cubes and a finite set of relations. In order to express user queries, relational algebra expressions are then extended to serve as a query language on grouping relations.

#### 2.2.1.2 Model of Gyssens and Lakshmanan

Gyssens and Lakshmanan [3] introduced a conceptual multidimensional data model for OLAP applications. The proposed model is a clear separation between structural aspects and contents. They propose data cube operator in a very elegant style. The algebra and calculus are equivalently presented for their model.

A multidimensional schema is defined as follows: Let *N* be a set of names, *V* be a set of values. An n-dimensional table schema is a tuple <D,R,par> where

$D=\{d_1,\ldots,d_n\}$ is a set of dimension names,

$R=\{A_1,\ldots,A_n\}$ is a set of attributes, and

Par: $D \rightarrow 2^{\{A1,\ldots,Am\}}$, such that

**(i)** for all $i,j=1,\ldots,n$, $i \neq j$, $par(d_i) \cap par(d_j)=\emptyset$, and

(ii) $U_{d\in D}\ par(d) \subseteq R$

$Par(d_i)$ is denoted by $X_i$. Let $M = R - U_{1\leq i\leq n} X_i$

An instance of an n-dimensional table schema <D,R,par> is a set of n+1 finite relations of the form $rd_1(Tid,X_1),\ldots,rd_n(Tid,X_n)$, $rm(rd_1.Ti,\ldots,rd_n.Tid,M)$ such that

- the join $¶_{Tid}(rd_1) \times\ldots \times ¶_{Tid}(rd_n)$ equals $¶_{rd1.Tid,\ldots,rdn.tid}$ (rm), i.e., for every combination of Tid values in the relations $rd_1,\ldots,rd_n$, there is at least one corresponding tuple in rm (fact table), and every tuple in rm corresponds to some combinations of Tid values in the relations $rd_1,\ldots,rd_n$;
- for all $i=1,\ldots n$, Tid is a key of the relation $rd_i$; and
- for all $i,j = 1,\ldots,n$, $i \neq j$, $¶_{Tid}(rd_i) \cap ¶_{Tid}(rd_j) =\emptyset$,i.e., the Tid values in different relations $rd_i$ and $rd_j$ are disjoint.

Multidimensional databases are considered to be a set of tables forming denormalized star schema. Attribute hierarchies are modeled through the introduction of functional dependencies in the attributes of dimension tables.

### 2.2.2 Models Based on Cube-Oriented

#### 2.2.2.1 Model of Agrawal, Gupta and Sarawagi

Agrawal et al. [1] proposed a pragmatic multidimensional model based on the notion of multidimensional cube including an algebraic query language. In their logical model, data is organized in one or more hypercubes. A cube consists of:

- *k* dimensions, and for each dimension name $D_i$, a domain $dom_i$ from which values are taken.

- Elements defined as mapping E ( C ) from $dom_1 \times \ldots \times dom_k$ to either an n-tuple, 0, or 1. E ( C )($d_1,\ldots,d_k$) refers to the element at "position" $d_1,\ldots,d_k$ of cube C.

- Part of the metadata is an n-tuple of names where each of the names describes one of the members of an n-tuple element of the cube. If the cube has no n-tuple element, then this description is an empty tuple.

An element of cell values can be either 0, 1, or an n-tuple <X1,…,Xn>. A cell containing "0" means that a combination of dimension values does not exist. A "1" represents the existence of that particular combination. An n-tuple indicates that additional information is available for that combination of dimension values.

#### 2.2.2.2 Model of Cabbilbo and Torlone

Cabbibo and Torlone [2] proposed a framework based on a logical data model for the design of multidimensional databases. The multidimensional data model is defined by the notations of dimensions and f-tables. Dimensions are formed in the DAG structure as the hierarchies of dimension levels, whereas f-tables are relations that contain a tuple of each cell of the data cube that contains numeric values.

A dimension schema is defined as a tuple <L, $\leq$, R-UP>. L is a finite set of dimension levels which is partially ordered by the relation $\leq$. R-UP represents the roll-up functions mapping the lower dimension level domain to higher dimension level.

A multidimensional schema is defined as a tuple <D,F> where D is a finite set of dimensions and F is a finite set of f-tables over these dimensions.

#### 2.2.2.3 Model of Vassiliadis

Vassiliadis [25] proposed a logical model for multidimensional database. The approach is based on the notion of the base cube.

The paper also provides a mapping scheme of the proposed multidimensional model to the relational model and multidimensional arrays.

A dimension is defined as a lattice $<H, \leq>$. $H = \{DL_1,\ldots, DL_n\}$ is a set of levels with a domain $dom(DL_i)$ attached to each level $DL_i$. The relation $\leq$ indicates a partial order on the dimension levels. Each dimension path is considered as a linear pattern. And each dimension contains a set of dimension paths.

The paper focuses on the modeling on cube base, and defines a basic cube $C_b$ as a tuple $<D_b, L_b, R_b>$ where:

- $D_b = <D_1, D_2,\ldots D_n, M>$ ; $D_i$ is the list of dimensions; M is a dimension that represents the measure of the cube.
- $L_b = <DL_{b1},DL_{b2},\ldots,DL_{bn}, ML>$ ; $DL_i$ is the list of dimension levels**;** ML is the dimension level of the measure of the cube.
- $R_b$ is a set of cell data containing the tuples of the data cube.

From the base cube, further cube can be derived by a set of operations. A cube **C** is thus defined as tuple $<D, L, C_b,R>$, where $C_b$ is the base cube.

### 2.2.3 Models Based on Object-Oriented

#### 2.2.3.1 Model of Trujillo and Palomar

In [7], the first revolutionary OO approach to MDB conceptual modeling is proposed. The Object-oriented multidimensional data model (OOMD) is defined by the two basic elements: dimension classes and fact classes. Then cube class (constructed from dimension classes and fact classes) is also presented to allow a user to accomplish a subsequent data analysis.

**Dimension Class (DC)** is a tuple of (A, ARR, E), where,

- A = Key Attribute (KA of Dimension Class) $\cup$ Dimension Attribute (DA)
- ARR is a possible attribute roll-up relation defined on A′, where $A' \subseteq A$
- E is the set of events allowed on the class objects.

**Fact Class (FC)** is constructed from n dimension classes and consists of (A, ARR, E), where,

- A = Key Attribute (KA of Fact Class) $\cup$ Fact Attribute (FA)
- ARR is a possible attribute roll-up relation defined on a subset of FA, where $FA' \subseteq FA$
- E is the set of events allowed on the class objects.

**Cube Class(CC)** is defined as a tuple (DC, FC, A, C, E, CE), where,

- DC is the set of dimension classes that have been used to construct the fact class
- FC is the fact class from which the cube class has been built
- A = KA $\cup$ CFA $\cup$ CDA where,
  - o KA is the set of key attributes of the FC
  - o CFA is a subset of FA from the FC
  - o CDA is a subset of the DA from $DC_i$

- C is a condition n-tuple ($a_1 = v_1$, $a_2 = v_2$,…, $a_n$=$v_n$) where $a_i$ are dimension attributes and $v_i$ the set of values that must fulfill each attribute $a_i$ to select the objects that will integrate this CC
- E is the set of operations allowed on the objects of the CC
- CE is the set of events (operations) permitted on the cube class.

This approach provides a higher level of abstraction (encapsulates both data and operations in one structure) especially; the model provides the classification hierarchy on attributes along dimensions by introducing ARR and domain functions to achieve the full semantic in the real world of a multidimensional case.

#### 2.2.3.2 Model of Nguyen, Tjoa and Wagner

In [4], a conceptual multidimensional data model is introduced. The model facilitates a sophisticated constructs of MD structure based on multidimensional data units or members such as dimension members, measure data values and then cells. The model is able to represent and capture natural hierarchical relationships among complexity of dimension members. The very suitable manners of definitions of three cube operators, namely, jumping, rollingUp and drillingDown are also presented.

In the approach, a multidimensional data model is constructed based on a set of dimensions $D=\{D_1,\ldots,D_x\} x \in \mathbf{N}$ , a set of measures $M=\{M_1,\ldots M_y\}$ $y \in \mathbf{N}$ and a set of data cubes $C=\{C_1,\ldots C_z\}$ $z \in \mathbf{N}$.

The model also presents the complexity of the dimensional structures, such as unbalanced and multi-hierarchical structures. Moreover, the data model represents the relationships between dimension members and measure data values by means of cube cells. As a consequence, data cubes and their operators are formally introduced. In addition, the authors proposed a modeling of the conceptual multidimensional data model in term of classes by means of UML, which is an object-oriented standard analysis and design notation.

## 2.3 Related Works

This section presents some works related to our research. Our research mainly focuses on the multidimensional database evolution and its solution is carried out through the Object-oriented paradigm. Therefore, only evolution approaches in object-oriented and multidimensional database including the multidimensional data model are discussed. The approaches of view maintenance [10, 21] are not considered.

As a result, this section emphasizes three related approaches, namely, Multidimensional Data Model, Schema Versioning Approach in Object-oriented Databases, and Schema Evolution in Multidimensional Databases.

### 2.3.1 Multidimensional Data Model

We have firstly discussed the research work of multidimensional modeling, which is a required concept of pursuance in schema evolution issue. In the modeling aspect, the new approach of OLAP modeling, namely, the object-oriented OLAP is taken into our consideration. The approach of Trujillo and Palomar [7] is adopted to be a dominant framework for defining the schema of our research. Trujillo and Palomar take the concepts and basic idea of the classical multidimensional model to propose a revolutionary approach based on the object-oriented (OO) paradigm to MDB conceptual modeling. Then, the basic elements of their Object-Oriented Multidimensional Model such as dimension classes and fact classes are introduced. They then present cube classes as the basic structure to allow a subsequent analysis of the data stored in the system. This approach is very applicable to the MDB schema in terms of dimension class and fact class. Therefore, we consider this modeling concept as a major structure of the MDB schema and extend by adding some key attributes suitable for the evolution analysis.

### 2.3.2 Schema Versioning Approach in Object-oriented Databases

In this section, we discuss an evolution technique used in our subject schema: OOMDB. By this, we adopt the schema versioning approach as a practical framework to our OOMDB schema.

Simon Monk and Ian Sommerville [12] proposed a model for class versioning in an object-oriented database. By defining update and backdate functions on attributes of the previous and current version of a class definition, instances of any version of the class can be converted to instances of any other version. This allows programs written to access an old version of the schema to still use data created in the format of the changed schema.

Another work belonging to Sven-Eric Lautemann [13, 14] is a candidate work for our interest. They indicate a way of handling the versioning history of a schema by specifying versions and relationships between versions by using schema derivation DAG (directed acyclic graph). Any schema version is traced by routing via DAG structure and relies on the defined conversion functions. Also, the author proposed a schema versioning approach which supports the dynamic change of an object-oriented database schema while it is used by running applications. Their mechanism allows applications to work with different schema versions based on the same single database.

In fact, the schema versioning approaches in object-oriented databases cannot be directly applied to the multidimensional case because of the characteristic of OLAP data analysis operations and the relationship between several dimensions and facts. In our research, we therefore develop the schema versioning algorithms that regard the whole schema change as well as the dimension and fact characteristics. Consequently, whenever changes occur, the schema version will be created and stored in the DAG structure.

### 2.3.3 Schema Evolution in Multidimensional Databases

Based on the object-oriented multidimensional schema and schema evolution in object-oriented database, we will then scrutinize the schema evolution issue in the existing research works of multidimensional database.

Blaschka [8] proposed a formal framework to describe evolutions of multidimensional schemas and their effects on the schema and the instances. The formal framework, FIESTA, is based on a formal conceptual description of a multidimensional schema and corresponding schema evolution algebra. Thus, the approach is independent of the actual implementation. The author also provided the set of evolution operations and described the effects of these evolution operations on the MD schema and the instances.

FIESTA formally defined what types of modifications occur on a multidimensional data model which is visualized by an ME/R model. Therefore, the schema evolution operations are dedicated for change via the graphical conceptual level. Here, the evolution of the structure of the dimension hierarchy (e.g., insert/delete dimension level), the evolution of classification hierarchy and evolution of fact are regarded. Thus, the framework reflects the evolution of multidimensional schema in a well-designed manner, however, the approach does not handle modifications of instances.

Nevertheless, we adopt some evolution operations from FIESTA that are suitable for our designed model as well as the representation scheme of the logical effects of the MDB evolution operations in terms of MDB evolution algebra. In our approach, the evolution of MDB schema and the instances are considered.

Another recent approach is the temporal approach of Eder and Koncilia [17]. This approach supports the change of dimension data by using the versioning technique. It is based on a temporal MD data model which not only a temporal version of dimension data, but also mapping scheme between different temporal versions of the instances of dimensions is regarded. In addition, a transformation matrix and mapping function are proposed for controlling the instance propagation between the schema version transitions.

However, this approach mainly focuses on the support of the evolution on dimension data, but not fully regards on the dimensional schema (i.e., dimension level). In addition, the proposed model is fit only for the database designed in temporal environment and it does not take the complex multidimensional structure into account.

Finally, we use the version of instances and version mapping approach of Eder and Koncilia as a guideline of our research. In particular, our research will concentrate on the modification of both dimensions and facts which are the basic elements of MDB schema. Furthermore, the effect of the schema update to the instance level will be considered. It is necessary to check the consistency of data for each schema version. Hence, if the schema change affects the dimension data or fact data (measure), the previous data will be retained as an instance version corresponding to the schema version. And the mapping scheme between schema versions and respective instance is provided for answering the query correctly. The versioning approach therefore fulfills our research concerns.

# CHAPTER 3

# PROBLEM STATEMENT

In this chapter, we describe the problems of the existing approaches associated with the subject of this research. Next, we state the research motivations as to why a support of multidimensional database schema is necessary. Hereafter, the research objectives and research contributions are articulated, respectively.

## 3.1 Problem of Existing Approaches

Considering the problems of the existing approaches of evolution issues in data warehouse or multidimensional database, we categorize the problems into three issues. These issues are described next.

### 3.1.1 Problem of View Maintenance Approach

In general, the warehouse database consists of materialized view over the operational data sources. When data is updated in the data sources, the view has to be updated accordingly [8]. Many research works proposed the idea on maintaining these views. Most of them concentrate on the incremental view maintenance technique which is beneficial for avoiding the full computation of the materialized view. In addition, the technique of self-maintainability of view has been deployed as another approach for evolution support. However, these approaches focus on maintaining the warehouse data when the data source has been changed and not maintaining the structural schema and data of the legal warehouse database. The view maintenance may yield other problems related to update anomalies such as view synchronization and consistency maintenance.

### 3.1.2 Existing Solution for the Evolution of Multidimensional Structure

Typically, there are two major approaches of the schema evolution in multidimensional database. The first one deals with the method in mapping data into the most recent analysis structure. In this case, the research focuses on updating multidimensional models. These models provide a pragmatic way for handling evolutions by mapping data in the latest version. But in this case, some data may be corrupted or even lost, for example, when a member is deleted. Since it is no versioning support, the schema modification is performed only with the latest version and thus the existence of evolution and information that may be critical for data analysis are hidden. In addition, the incorrect result (false) may occur from analysis that does not take into account the explicit evolution of information over time.

The second approach refers to the tracking history solution. In fact, very few research focused on this issue. Some approaches [26, 27] choose to represent data in the temporal consistent mode of presentation corresponding to an exact view. However, most works relied on the MD traditional model, e.g., relational model or cube model (which will be stated in the next section). Recently, Eder and Koncilia [9] have proposed a mapping function to link between structure versions. But, it provides a partial solution, in which the schema evolution and time consistent presentation are not taken into account. Moreover, the consideration of complex dimension structures is not served as well.

### 3.1.3 A Multidimensional Modeling Issue

To scrutinize the subject of schema evolution, the data model is required. However, the MD model used by the recent approaches [4, 7, 8] relies on the traditional MD cube model or relational model, which have some drawbacks such as flexibility for implementation and semantic power of the complex MD structure. As a result, this problem is one of the current research issues, and this aspect highly inspires our research work to pursue on another powerful multidimensional data modeling based on object-oriented modeling.

## 3.2 Research Motivations

It is widely recognized that data warehousing and a multidimensional database system or OLAP are essential for strategic decision support. To construct a reliable data warehouse, related data sources are extracted, transformed, cleansed and integrated to a dedicated data warehouse. Typically, the data warehouse is modeled in terms of multidimensional view consisting of multidimensional axis that truly reflects the perspective of user's needs. This base is commonly referred to multidimensional database. Even after a multidimensional database system has been deployed, changes of schema still occur, since the OLAP user works directly with the multidimensional schema. This is contrary to traditional applications where the user works with an application program encapsulating the schema details of a relational or object-oriented database system [8]. In OLAP system, the users may state the changes of their requirement any time because they work with the different dimensions and fact which are the significant components constituting the MDB schema. This implies the structural changes of the database schema. Nevertheless, a complete methodology for maintaining the actual multidimensional database schema has only few works in the research community. Owing to the complexity of the evolution of MDB schema, it thus inspires our research vision.

Actually, the research issue related to the schema evolution support is well studied and published in the object-oriented database area. In OO context, the schema evolution involves the support of complex representation of class and object characteristics incorporating with its properties such as inheritance, aggregation/decomposition, generalization/specification, etc. As there is a need to support the changes of the complexity of multidimensional structure, it is highly motivated to apply the object-oriented schema evolution technique to the multidimensional case. As a result, this implies that there is also a need to use the object-oriented framework to model the object-oriented multidimensional model to support its schema evolution technique.

## 3.3 Research Objectives:

1. To identify the problems associated with schema evolution in multidimensional database systems.

2. To propose an alternative framework based on schema versioning approach used in object-oriented paradigm for schema evolution support in multidimensional database.

3. To define and develop the MDB schema evolution algorithms as a major mechanism in conducting the schema evolution and retrieval process.

4. To develop a schema evolution manager prototype as an easy-to-use tool for maintaining and controlling the schema update in a multidimensional database system.

## 3.4 Scope of the Research

This thesis is subject to the multidimensional database schema evolution support. We formally develop a methodology for supporting the evolution of MDB schema based on OO model and adopt the schema versioning technique used in OO approach to apply to OOMDB schema as well. Equally important, the MDB schema evolution operations and its own descriptive algebra are also provided. Actually, our approach mainly focuses on the change of the MDB schema level. However, thanks to the schema change, the evolution effect may lead to instance adaptation. Practically, our approach not only supports the intuitive change of the schema level but also the propagation of instance initiated from the schema modification is supported. In addition, the prototype called OOMDB SEMAN (Object-Oriented Multidimensional Database Schema Evolution Manager) is developed as a supported tool for multidimensional schema update.

## 3.5 Research Contributions:

The main contributions of this research are as follows:

- The development of an object-oriented multidimensional data model (OOMD). This OOMD model is designed to serve for our MDB versioning approach specifically. Technically, the proposed model represents the complex structure of multidimensional context, and later it will be used to construct the schema versioning model in a suitable way.

- The development of an alternative methodology for multidimensional database schema evolution support. We formally develop a methodology for supporting the evolution of MDB schema by applying the schema versioning technique used in OO approach to the multidimensional database system. Conceptually, the versioning approach proved to effectively support the change of MDB schema in both the structure and instance level.

- The analysis of schema evolution effects when the structural change has been triggered. Syntactically, we provide the semantic definitions of the evolution operations as well as its evolution effect in terms of an understandable descriptive algebra.

- The development of the schema evolution manager system prototype. We test our proposed idea by developing a system prototype called OOMDB SEMAN (Object-Oriented Multidimensional Database Schema Evolution Manager) as an administrative tool to support the entire process of MD schema changes. OOMDB SEMAN also allows the OLAP designer to perform schema modification via its friendly interface in an effective manner.

# CHAPTER 4

# AN OBJECT-ORIENTED VERSIONING APPROACH FOR MULTIDIMENSIONAL DATABASE SCHEMA EVOLUTION

## 4.1 The Overall Framework of the MD Schema Evolution

Basically, this research solution is elaborated from three key concepts: OOMDB schema, schema versioning approach and multidimensional schema evolution characterized according to FIESTA approach [8]. Figure 4.1 draws the idea of the conceptual view of our MD Schema Evolution Framework.

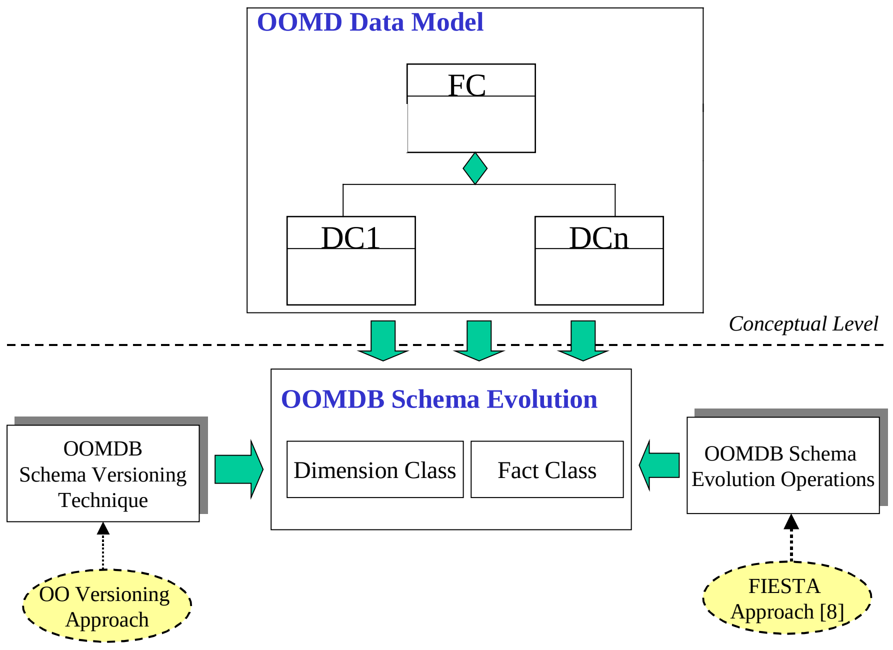


**Figure 4.1 Conceptual View of MD Schema Evolution Framework**

In the conceptual level, the multidimensional model is constructed in the form of Object-Oriented Multidimensional Model (OOMD) used to represent the major structure of multidimensional database (MDB). The OOMD model is mapped to OOMDB schema consisting of dimension classes and fact class. Additionally, the

OOMDB schema is an integrated center of the schema versioning approach and MD evolution characteristics. Owing to a high degree of flexibility in implementation and popularity of relational databases, our OOMDB schema is built on top of the relational system. We go through each component of the framework in a deep concern in the next sections.

## 4.2 Multidimensional Data Model

To scrutinize the subject of schema evolution, the data model is a prerequisite background that needs to be provided. In our framework, the MD schema is an innovative application of the object-oriented multidimensional model (OOMD) [7,19] to the model of Blaschka[8]. Therefore, the extended model is designed to fit our thesis's core with a special focus on comprehensive and understandable definition.

**Definition 4.2.1: OOMDB Schema:**

**Dimension Class(DC)** is defined as a tuple DC<KA, DA, DL, ARR>
where:

- KA is a key attribute for the class e.g., product_id
- DA is a set of dimension attributes that is not considered to be a dimension level but constitutes the dimension class.
- DL is a set of dimension levels
- ARR represents the order of dimension level (l) that states the partial order for Roll up function: $R_{up}\ l_1 \text{->} l_2$ .

An attribute roll-up relation (ARR) is an n-tuple $<l_1,\ldots,l_n>$ where a partial order relation is defined, such that $l_1<l_2<\ldots<l_n$ and that given two $l_i$, $l_j$ such that $a_i<a_j$, there is not any $a_k$ such that $a_i<a_k<a_j$

**Note** that ARR for each dimension class may contain several R-up functions, which means there are several possible hierarchy relationships for the dimensional schema. For example, ARR in Time dimension consists of two hierarchical relationships for their dimensional structure: $R_{up1}$ (day $\rightarrow$ month $\rightarrow$ year), $R_{up2}$ (day $\rightarrow$ week $\rightarrow$ year)

**Fact Class** is defined as a tuple of <KA, Gran, FA>
where:

- KA is a key attribute for fact
- Gran is a finite set of base dimension level names (represented by its dimension class name) associated with the fact class
- FA is a set of fact attributes or measures

**An OOMDB Schema S** is a tuple <D,F> where D is a finite set of dimension classes and F is the fact class constructed from these dimension classes.

**Example:** we use the MD Schema example taken from [29]. A grocery database is constructed in multidimensional structure consisting of 4 dimensions: *product, time, promotion, store* and single fact *sales*.

Figure 4.2 displays the schema of the cube sale constructed by 4 dimensions: product, promotion, time and store.

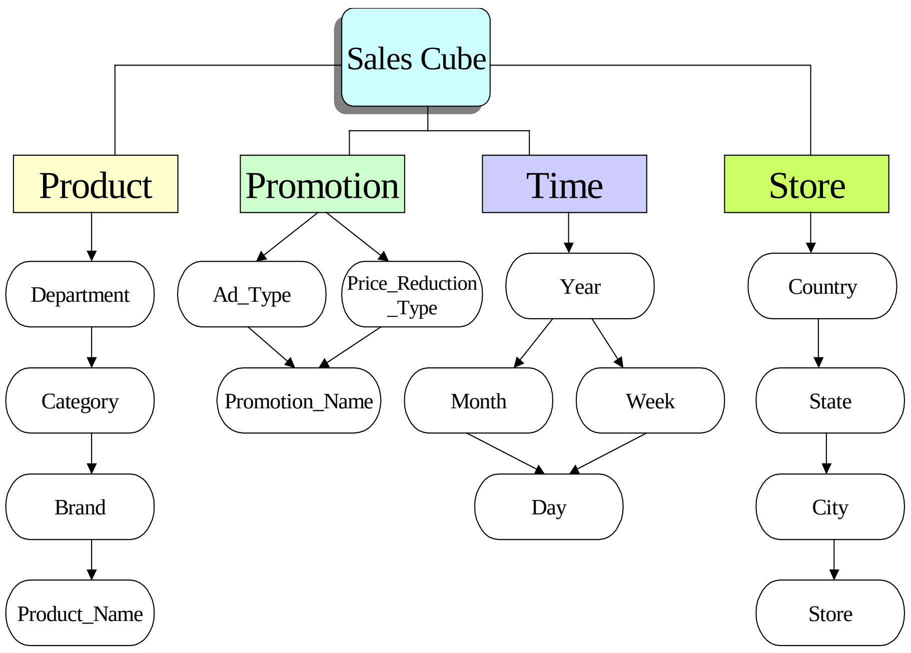


**Figure 4.2: The schema of the cube sale, constructed by four dimensions: Product, Promotion, Time and Store**

The example MD Schema *S* <D, F> has the following components:
D = {Product, Promotion, Time, Store}
F = {Sales}

Each DC contains the following components:

DC$_1$: Product (**KA**:Product_id, **DA**:{ package_size, weight}, **DL**: {Product_Name, Brand, Category, Department}, **ARR**: {product_name $\rightarrow$Brand $\rightarrow$ category $\rightarrow$ department})

DC$_2$: Promotion (**KA**: Promotion_id, **DA**: {}, **DL**: {Promotion_Name, Ad_Type, Price_Reduction_Type}, **ARR**: {(Promotion_Name $\rightarrow$ Ad_Type),(Promotion_Name $\rightarrow$ Price_Reduction_Type)})

DC$_3$: Time (**KA**: Time_id, **DA**: {}, **DL**:{ Date, Week, Month, Year}, **ARR**: {(Date $\rightarrow$ Month$\rightarrow$ Year), (Day $\rightarrow$ Week $\rightarrow$ Year)})

DC$_4$: Store (**KA**: Store_id, **DA**: {Store_District},**DL**: {Store_Name, City, Country, State}, **ARR**: {Store_Name $\rightarrow$ City $\rightarrow$ State$\rightarrow$ Country})

**FC** contains the following components:

**FC**: Sales(**KA**: Sales_ID, **Gran**: {product, promotion, time, store), **FA**: {dollar_sales, dollar_costs, unit_sales}

After having defined the MD schema, this section presents the definitions of the MD instances in terms of its definitive domain.

**Definition 1: Domain of a Dimension Level**

The domain of a dimension level l $\in$ L where L is a finite set of the level associated with the fact class.

dom(L) = {(DL)m$_1$,….(DL)m$_n$} where M are dimension member names.

Let us assume the following domains for the base levels of each dimension class:

dom(product_name) = {“beef stew”, “chicken dinner”, “lots of nuts”,…};

dom(promotion_name)={" Blue Ribbon Discounts", "Big Pro",…};
dom(day) = {"01/01/2000", " 01/02/2000",…};
dom(store_name) = {"Store No.1", "Store No. 2",...}

**Definition 2: Domain of a Dimension and Fact Attributes**

Let A be a set of attributes being hold by the object of a particular class (dimension class or fact class)

The Dimension Attribute (DA) is defined as an n-tuple $(a_1,\ldots,a_n)$ that characterizes the dimension objects of the dimension class. However, the Dimension Attribute is not a subject contained in the cube class that is used for subsequent data analysis.

The Fact Attribute (FA) is defined as an n-tuple $(a_1,\ldots,a_n)$ that characterizes the fact objects of the fact class and the FA is permitted to constitute the cube cell.

For example,
DA = {package_type, weight, store_district}
FA = {dollar_sales, dollar_costs, unit_sales}

The domain of the attribute A is considered by means of domain function **dom: $a_i \rightarrow v_i$** where $a_i \in A$ and $v_i$ are the set of possible values (instances) taken by $a_i$.

**Definition 3: Domain of Fact**

The domain of a fact is the cross-product of all base dimension levels (representing the coordinates of the cube cell) which are associated with the fact class and all combinations are mapping to fact attribute (FA) that will be characterized as a fact object or measure of the cube class.

$$\mathrm{dom}(f) := \underset{l \in \mathrm{gran}(f)}{\mathbf{X}} \mathrm{dom}(l) \rightarrow a|\mathrm{attr}(a) = FA(f)$$

## 4.3 OOMDB Relationship and Object Behavior

In addition to the basic construct of OOMDB incorporating of dimension classes and fact class, there are two important elements for the underlying schema.

### 4.3.1 The Association Relationship between Classes.

Basically, the relationship between fact class and dimension classes is mapped in respect to super class and sub-class relationship, i.e., the fact class is a super class of several associated dimension classes.

Generally, the relationship between fact class and dimension classes is denoted as many-to-many relationship.

### 4.3.2 Object Behavior or Method Contained in Each Class.

We classify methods of OOMDB into two groups: basic OLAP method and evolution method. The basic OLAP method is the normal property or operation held by each class, e.g., slice, dice, roll-up, drill-down, while the other is the method that affects the change of OOMDB schema, which is defined in section 4.5.

Since multidimensional queries are not within our scope, we do not state multidimensional operations or instance operations here. In contrast, the operations that work with the multidimensional schema are defined. These schema evolution operations are discussed in section 4.5.

## 4.4 MDB Schema Integrity Constraints

The following rules are defined as two basic integrity constraints.

1. Fact Class must be connected to at least one dimension class
2. The name of fact class, dimension class, fact attribute, dimension attributes and dimension levels are all disjoint, i.e., $FC \cap DC \cap FA \cap DA \cap DL = \varnothing$

## 4.5 A Schema Versioning Technique for MDB Schema Evolution

A schema versioning approach plays a major role in the framework. Even though the versioning technique is very well applicable in object-oriented paradigm, we cannot apply this technique to our OOMDB schema directly because of the complexity of MDB's characteristics. Consequently, the versioning methodology is adjusted to be suitable for our defined schema.

We introduce the operational semantics of each schema change in terms of MD evolution operations. In fact, schema changes do not only affect schema but also data. As to the underlying database, our model also deals with the change propagation problem. The defined MD operational semantics tell how a legal dimensional data or factual data with respect to a schema version should be modified in order to ensure consistency after the schema version modifications.

In our approach, the version of MD schema is generated once the change of schema is detected. The consequence of schema changes is performed by our proposed functions: schema version transformation function (*SVT)* and conversion function (C*f)*. Also, we define the transformation algorithm to carry out the evolution process. Actually, a version of an object can be seen as a snapshot of this object taken at a certain time. A schema derivation DAG (directed acyclic graph) is used to express the relationship between all versions of schema and the respective dimension and a fact object.

To manage the schema versions efficiently, we introduce the schema versioning model based on the proposed MD schema.

### 4.5.1 Schema Version Model

**Schema Version (SV)** is a 4-tuple <SVid, [Ts,Te], D, F>

- SVid = Schema Version ID
- [Ts,Te] is a period of time(time start and time end) of the schema version
- D is a set of dimension classes contained in the MDB

- F is a fact class associated with the set of dimension classes.

### 4.5.2 Applying Schema Changes to MD Schema and Instance Propagation

Intuitively, the schema change affects a schema version and produces a new one. In most cases, such changes have to be propagated to instances in order to ensure the consistency with respect to the new schema version.

Formally, we introduce two functions, *Schema Version Transformation* and *Conversion Function*. The first function is used to specify the change of structural MD schema from the preceding version to a later new version, whereas the second function is used to control the propagation of MD instance.

**Definition 4.5.2: Schema Version Transformation Function, SVT**

The Schema Version Transformation is a function that maps the old structural schema to a newer version. Let ***O*** be the set of all possible schema evolution operators affecting the change of MD schema, and $SV_u$ be a preceding schema version of a modified schema versions $SV_v$.

**$SVT_O(u,v)$** is the function used to transform the structural schema of $SV_u$ to $SV_v$ with the operator ***O,*** where **u,v** $\in$ ***N*** , and ***N*** is the closure set of versions contained in DAG

The mapping domain of the function is defined as:

$$\mathbf{SVT{:}(SV_u \xrightarrow{O} SV_v)}$$

**Definition 4.5.3: Conversion Function, *Cf***

Conversion functions are implemented at the class level, i.e., they map a dimension instance or fact instance of one class version $SV_u$.C of a source schema version $SV_u$ to the destination schema version $SV_v$.C. The conversion function is applied to generate a copy of object versions of dimension classes or a fact class that have been changed according to the effects of the changes on the old to the new schema versions. Therefore, the original version of MD instance is not lost during the propagation process. In addition, the Data Management function is designed to work with the C*f* to maintain the incremental data for each schema version.

Let $SV_u$ and $SV_v$ be versions of MD Schema *S*, both containing a version of class ***C*** and $SV_u$ be a direct preceding version of $SV_v$.

$\boldsymbol{Cf_o(I_{u.c}, I_{v.c})}$ is the function used to control the mapping of object instance of the changed class C of $SV_u$ to a resulting class C of $SV_v$, where ***I*** is a set of object instances of the SV.

The mapping domain of the function is defined as:

$$\boldsymbol{Cf:(\, I_{SVu.c} \xrightarrow{o} I_{SVv.c})}$$

**Definition 4.5.3.1: Properties of *Cf***

Basic Step: To prove that class $C_i$ of $SV_u$ is converted correctly by
the *Cf*.

Let $C_i$ be a class of the $SV_u$ where c is the classes constituting the schema, and I be the set of instances of schema version.

**Case I**: $C_i$ of SVu has changed

$I_{SVu.c}$ --> $I_{SVv.c'}$

**Case II**: $C_i$ of $SV_u$ has not changed

$I_{SVu.c}$ --> $I_{SVv.c}$

**Induction Rule:**

As the effect of changes on the instances occurs in either case I or II, thus the mapping process of the instance of any schema version is properly controlled by *Cf* of each unique SV pair and it is originated from what we show in Basic Step.

Example: If $DC_1$ of $SV_u$ has changed, then the set of instances of each DC in $SV_v$ would be mapped as follows:

$I_{SVu.DC1}$ --> $I_{SVv.DC1'}$

$I_{SVu.DC2}$ --> $I_{SVv.DC2}$

……

$I_{SVu.DCn}$ --> $I_{SVv.DCn}$

Note that $I_{SV.FC}$ will be unchanged only if operations ***O*** are ADD DA and Delete DA. (List of operations O is presented in section 4.6)

Technically, the schema version transformation function (*SVT*) and conversion functions (*Cf*) indicate the conversion path between schema versions in DAG.

Although the support of consistent MD schema version is a core subject, the cost of storage of multi MD versions is also a concern in this research. We maintain a copy of object instance of the modified schema version by using the existing base class (dimension class and fact class). Thus the dimension data and factual data are retrieved and calculated immediately once the schema version is specified.

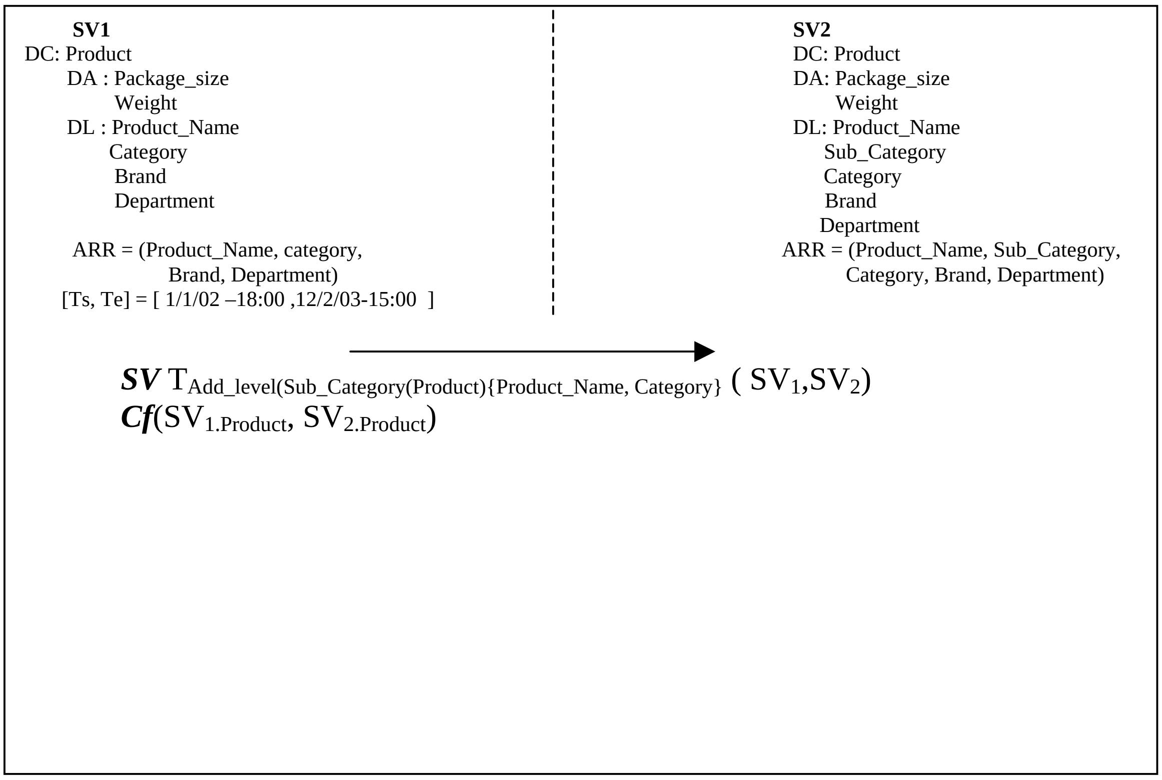
To ensure the correctness and the consistency of data, the object instances are regarded in terms of MD instances of the dimension class or fact class with respect to the MD schema version. Also, the schema version model has an attribute time [Ts, Te] for each schema version; therefore, the legal instance contained in the corresponding base dimension or fact class is always synchronously matched.

Occasionally, there might be the loss of instances when the MD schema evolution operations are applied, especially for a group of deletion command. This yields the loss of data for the modified version. In our approach, the set of deleted instances are always available in the base class (DC and FC) because these instances are not allowed to be physically deleted. In fact, changes of instance are recorded in terms of SQL statement.

Since we do not retain a duplicate copy of the object instance for each class but maintain from the individual base class, the cost of version storage is reduced. However, there exists a trade-off for the processing time in switching the schema version in which the base fact class needs to be recomputed. On this ground, the efficient OLAP indexing technique would be a potential key to address this concern in the future.

In order to understand our mechanism in applying schema versioning to well-defined multidimensional schema, we provide an example of schema changes in three cases, namely, add dimension level, delete dimension level, and add dimension class to fact class. The details of these cases are described as follows:

**Case1: Schema Changes by " Add Dimension Level"**



**SV1**
DC: Product
DA : Package_size
Weight
DL : Product_Name
Category
Brand
Department

ARR = (Product_Name, category, Brand, Department)
[Ts, Te] = [ 1/1/02 –18:00 ,12/2/03-15:00 ]

**SV2**
DC: Product
DA: Package_size
Weight
DL: Product_Name
Sub_Category
Category
Brand
Department
ARR = (Product_Name, Sub_Category, Category, Brand, Department)

$\boldsymbol{SV}\ T_{\overrightarrow{\text{Add\_level(Sub\_Category(Product)\{Product\_Name, Category\}}}}\ (SV_1, SV_2)$
$\boldsymbol{Cf}(SV_{1.\text{Product}}, SV_{2.\text{Product}})$

**Figure 4.3: Example of MD Schema Versioning (Add dimension level)**

Figure 4.3 reveals an example of MD schema versioning (Add dimension level). In this case, the "Sub_Category" is added as a new dimension level of the *Product* class. The effects are the evolution of the change of attribute roll-up relationship (ARR) of the product class, and the change of granularly of the fact class. Therefore, a new version $SV_2$ is added with a new dimension level including the updated classification relationship in its dimension schema, and also the fact class is updated.

### Case 2: Schema Changes by “ Delete Dimension Level”

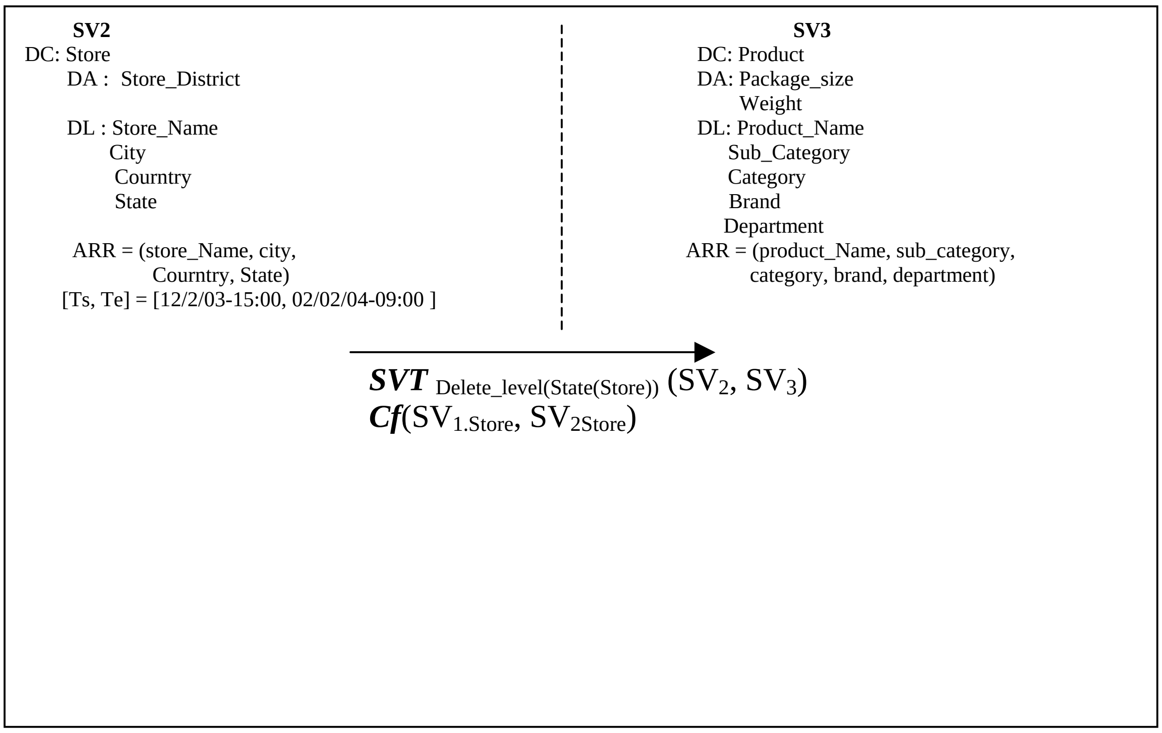

SV2
DC: Store
DA : Store_District
DL : Store_Name
City
Courntry
State
ARR = (store_Name, city, Courntry, State)
[Ts, Te] = [12/2/03-15:00, 02/02/04-09:00 ]
SV3
DC: Product
DA: Package_size
Weight
DL: Product_Name
Sub_Category
Category
Brand
Department
ARR = (product_Name, sub_category, category, brand, department)
$\boldsymbol{SVT}_{\text{Delete\_level(State(Store))}}(SV_2, SV_3)$
$\boldsymbol{Cf}(SV_{1.Store}, SV_{2Store})$

**Figure 4.4: Example of MD Schema Versioning (Delete dimension level)**

Figure 4.4 shows an example of MD schema versioning (delete dimension level). In this case, the dimension level “State” is deleted from of the *Store* class. Similar to the “Add dimension level” operation, the evolution effects occur in both of dimension class and fact class. For the dimension class, the attribute rollup relationship (ARR) of the store class are changed by eliminating the dimension level “State” from the classification relationship of its intra class, and the store level which is associated with the fact class is deleted.

**Case 3: Schema Changes by " Add Dimension Class to Fact Class"**

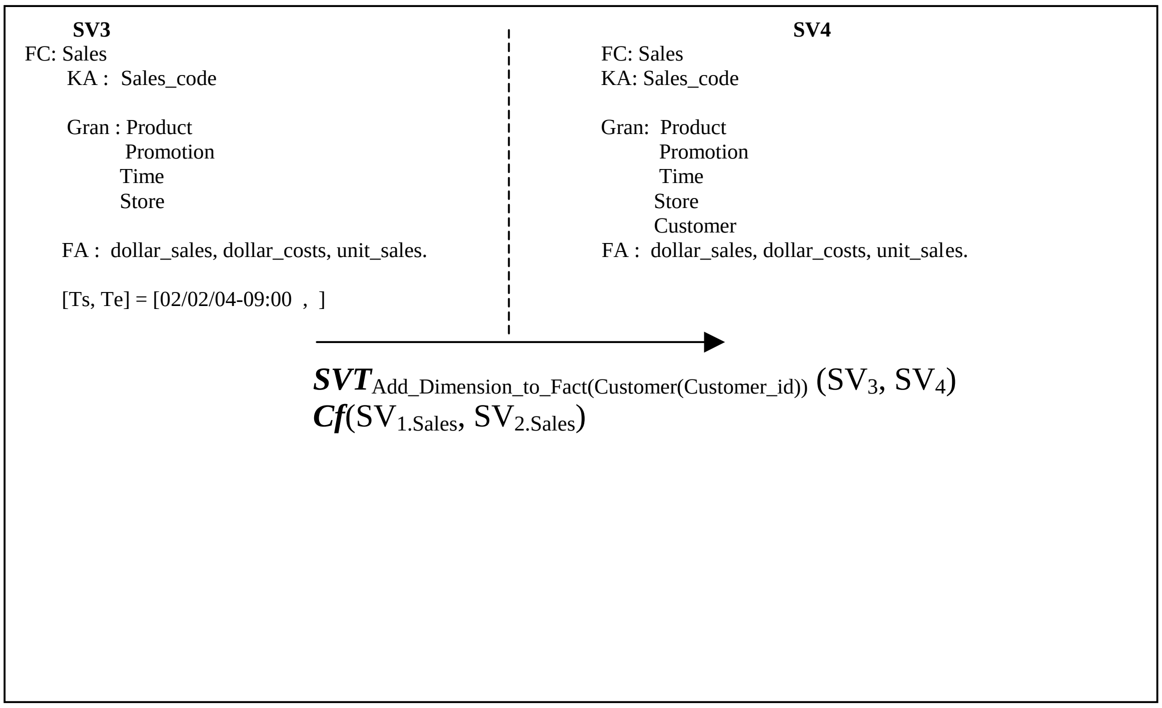

SV3
FC: Sales
KA : Sales_code
Gran : Product
Promotion
Time
Store
FA : dollar_sales, dollar_costs, unit_sales.
[Ts, Te] = [02/02/04-09:00 , ]
SV4
FC: Sales
KA: Sales_code
Gran: Product
Promotion
Time
Store
Customer
FA : dollar_sales, dollar_costs, unit_sales.
$\boldsymbol{SVT}_{\text{Add\_Dimension\_to\_Fact(Customer(Customer\_id))}}(SV_3, SV_4)$
$\boldsymbol{Cf}(SV_{1.Sales}, SV_{2.Sales})$

**Figure 4.5: Example of MD Schema Versioning (Add Dimension Class to Fact Class)**

Figure 4.5 displays an example of MD schema versioning (Add dimension class to fact class). In this case, a new dimension class "Customer" is added as a new association member of the fact class. By this, the sales fact class is updated in both its granularity elements which are added by a new set of DL of customer class, and the instance of the fact.

Note: For further evolution effects of other schema evolution operations, please see section 4.6.

**Rule 4.5.1: Schema Derivation**

The derivation of a new schema version SVv from existing SVu is processed by using exactly one Schema Version Transformation (*SVT*) and Conversion Function (C*f*) for each derived class *c*, as seen in DAG in Figure 4.6.

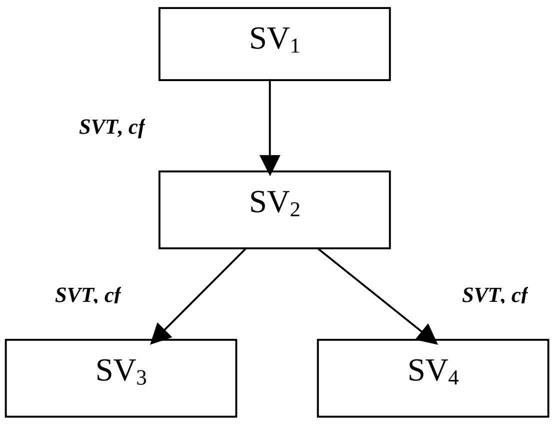


**Figure 4.6 Schema Derivation DAG**

Logically, the schema versions are formed in the way that the newer schema version must be generated from the existing historical version and the instance is propagated through the conversion function along the derivation path. Therefore, the order of schema version is significant for tracing the path.

**Rule 4.5.2: MD Schema Version Retrieval**

The retrieval process is performed by calling the available MD schema version in DAG i.e., we can retrieve any structural schema version stored in DAG. Technically, the set of legal instances is retrieved from the set of possible object values of the current base object classes and the set of added, updated or deleted instances derived from the execution result of SQL commands. These SQL records are generated when any instance of the object class has been changed. The idea of the MD schema version retrieval is detailed in section 4.4.4.2 (Backward Schema Version Compatibility Algorithm). As a result, the whole structural schema and the set of possible object values are always well preserved.

Therefore, in our framework the MD schema version is treated in a very flexible and conservative style to allow databases to evolve forward and convert the schema and its instance back to an earlier version.

**Definition 4.5.4: Well-defined DAG**

To determine the property of well-defined DAG, we adopt the criteria of Lautemann [13] for the consideration.

A schema derivation DAG <$SV_1$,…,$SV_n$) is well defined (with respect to schema version transformation and conversion functions), iff each pair of schema version $SV_x$, $SV_y$ $\in$ $SV_n$, ($x \neq y$) and for each class *C* in $SV_x$ and $SV_y$ in the following holds:

- Existence: a conversion path for class *C* exists from $SV_x$ to $SV_y$ and
- Uniqueness: the conversion path is unique.

**Lemma 4.5.1: Well- defined Transformation**

If each SV $\in$ $SV_n$ is derived from respective Rules 4.5.1 and 4.5.2, the resulting schema derivation represented by DAG is well defined (with respect to schema version transformation and conversion functions).

**Proof:** Lemma 4.5.1 is instantiated by induction on the set of MD schema versions $SV_n$ of the DAG following the historical order of their induction.

- $SV_{(n+1)}$ is derived from only one schema version $SV_u$
    - Existence: By induction assumption, structural schema can be transformed into $SV_{n+1}$ from $SV_n$ by the function $\boldsymbol{SVT_O}$($SV_n$ , $SV_{n+1}$) and SVn $\in$ DAG. Therefore, when the specific query requests for any schema version (SVn), that schema version will be processed and prompted to answer that query.

      Then, instances are propagated into SVj.c from each $SV_n$ via a conversion function $Cf_{c,j \rightarrow u}$ . By Rule 4.5.1, we have $Cf_{c,\, u \rightarrow n+1}$ and the conversion into $SV_{(n+1)}$.c can be done by $Cf\ c,_{j\rightarrow\ n+1:} = Cf_{\ c,\ u\rightarrow n+1\ O}$ $Cf\ c,_{j\rightarrow\ u}$

- Uniqueness: By Rule 4.5.1, Only ***SVT*** : $\boldsymbol{SVT_O}(SV_n\ , SV_{n+1})$ and $Cf_{\ c,\ n\rightarrow n+1}$ are added to the set of schema version transformation function and conversion function. As $SV_{(n+1)} \not\subseteq SV_n$, no cycles can be introduced by the derivation of $SV_{(n+1)}$.

### 4.5.3 Dealing with Different MD Schema Versions and Their Interschema Relationships

Formally, an evolving schema consists of a collection of schema versions, each of which is associated in terms of derivation path of DAG and the transforming relationship between the two connected schema versions is specified by *SVT* and *Cf* functions. To trace back to any schema versions, the structural schema version of dimension classes and a fact class could be retrieved from DAG and the legal instance (object value) could also be consistent with the corresponding transformed schema version.

Additionally, we introduce the versioning algorithms to provide forward and backward compatibility for MDB schema evolution. Therefore, the evolution of MDB schema and instance are well-performed. When the schema has been changed by evolution operations, it ensures the transformation of a schema version to a newer schema version without the problem of inconsistency and incompatibility. And if the user query requires factual data of the historical schema version, it is still available and prompt to answer the query by backtracking to the specified schema versions and the corresponding object value stored in DAG--schema version’s storage. Moreover, not only the consistency of MD schema evolution is concerned but the cost of version storage is also taken into account. In our approach, we provide the storage to maintain the structural schema of each version whereas the instances are well maintained by its respective base class and data management class. Thus, our idea demonstrates that the set of legal MD instances would be consistent with its schema version by means of the record of data manipulation. Technically, the changes of the MD schema are triggered from the schema evolution operations. In practice, the changes of structural schema may lead to the changes of instance as well. To control the change of the instance level, the data manipulation record is treated as the generated SQL statement of the particular schema version where its instance has been added, updated or deleted. By concept, the instance versions are thus kept in terms of the corresponding SQL statements. Therefore, the legal instance of the object version is constructed from the respective base class and the operation results of the SQL statement of the specified schema version. Accordingly, this versioning basis is thus a contribution point of our core of MDB schema evolution framework.

### 4.5.4 OOMDB Schema Versioning Algorithms:

There are two important algorithms to conduct the schema evolution and retrieval process. First, Forward Schema Version Compatibility Algorithm (FSV Algorithm) is used to support database schema transformation which is triggered by the evolution operations, or the algorithm is used to support the property of forward compatibility. The other algorithm is  Backward Schema Version Compatibility Algorithm (BSV Algorithm). The BSV algorithm is required to ensure the availability of the historical schema versions, which is the important concept of the versioning approach; the property of backward compatibility is therefore also supported.

#### 4.5.4.1 Forward Schema Version Compatibility Algorithm:

The idea of this algorithm is that after a schema version $SV_u$ is selected, the schema transformation process starts with the schema updates required to change $SV_u$ into $SV_v$ . By concept, the algorithm would first check the syntax of all possible MD schema evolution operations and also verify the operation type in order to propagate the change in the structural schema and instance level.  When the transformation process is completed, the new schema version $SV_v$ is included into the schema derivation DAG as a child of $SV_u$.

1. Check syntax of operations

```
         Add_level((new level l (DL),{$l_{new-1}$, $l_{new+1}$}; add dimension level
         Delete_level((level l (DL) ; delete dimension level
         Add_attribute ((new attribute $a_{new}$ (DC) ; add dimension attribute
If O =   Delete_attribute ((del attribute $a_{del}$ (DC) ; delete dimension attribute
         Add_dimension_into_fact((KA(DC),L, FC; add dimension class to fact
         Delete dimension_from_fact ((KA ($DC_{del}$), FC; delete dimension class _ from fact
```

Then perform the operations

2. Perform the schema change based on the evolution operation *O*

```
If O = Add_level((new level l (DL),{$l_{new-1}$, $l_{new+1}$}; add dimension level
      OR
      Add_attribute ((new attribute $a_{new}$ (DC) ; add dimension attribute
   Then Add DA or DL and its legal instance at the specified base dimension class
Elseif
   O = Delete_level((level l (DL) ; delete dimension level
      OR
      Delete_attribute ((del attribute $a_{del}$ (DC) ; delete dimension attribute
   Then Drop the specified DA or DL at the prompted versions solely
Elseif
   O = Add_dimension_into_fact((KA(DC),L, FC; add dimension class to fact
      OR
      Delete_dimension_from_fact ((KA ($DC_{del}$), FC; delete dimension class from fact
   Then Fact class needs to be recomputed according to the associative DCs.
   "Recompute Fact domain"
Endif
```

3. Record the version of schema

   3.1 Record the actual schema before change as <SVid, [Ts,Te], D, F>

   3.2 Assign Version number followed by the former version in DAG

4. Transform the change (by means of *SVT* and *Cf* functions) to a new schema version and assign time_start [$T_s$] to a new derived schema version.

### 4.5.4.2 Backward Schema Version Compatibility Algorithm

This algorithm is employed when the database schema is requested for the historical schema versions. Logically, the algorithm checks the schema version number requested by user and retrieves a schema version as a snapshot from the DAG. Then, the instance of each dimension class is invoked as a view by composing a set of instances of current base dimension classes and a fact class. In addition, the algorithm then checks the data manipulation operation logged in Data Management class to see whether or not the specified schema version has any changes in the instance level. If the change of multidimensional instances of the specified schema version is detected, the corresponding SQL statement will be executed.

1. Check the schema version specified by a user
2. Retrieve the structural version of the schema from DAG.
3. Retrieve the instance of the specified schema version from the base DC
4. Check SQL statements in Data Management Class. If the SQL record of the specified version is detected, the SQL statement will be performed against the set of dimension and fact data.
5. Check interval valid time of the specified schema version
6. Re-compute fact class from all possible dimension classes (of the specified schema version) and its measure.
7. Get the specified schema version and the legal cube to answer the query.

## 4.6 Evolution of MDB Schema

The set of MDB schema evolution operations is adopted from FIESTA [8], however, we leave some operations that are not suitable for our schema.

In order to easily explain the semantic and evolution effect of each operation, we borrow the representation scheme from FIESTA and present the MD evolution effect in an algebraic term.

### 4.6.1 Modification of a Dimension Level

1. **Add Dimension Level:** this operation extends a dimension class with an additional dimension level. The operation leads to a change of classification relationship within dimension class. For example, the addition of classification relationship is required. Table 4.1 shows Add dimension level operation.

| **Add dimension level** | |
|---|---|
| Syntax with input and output parameters | **Add_level**(new level ***l*** (DL), **I**{$l_{new}$ }) ) **Input**: Schema DC.$V_i$, instances I.$V_i$, new level name $l_{new}$ **Output:** new schema DC′.$V_j$, new instances I'$V_j$ |
| Pre-condition(s) | $l_{new} \notin$ DL,ARR |
| Post-condition(s) | $l_{new} \in$ DL, ARR |
| Example | Add_level(Semester(Time)), **I**{$l_{new}$ }) |
| Semantic expressed by means of the MD schema | **Schema:** S′; **DC** <KA, DA, DL $\cup$ {$l_{new}$},ARR′: ARR $\cup$ ( {$l_{new}$})> **FC** <KA, Gran′: Gran(*f*) $\cup$ $l_{new}$ , FA> |

**Table 4.1: Add Dimension Level Operation**

2. **Delete Dimension Level:** this operation deletes an existing dimension level from the dimension class, and the classification relationship operation is also performed. Instances are deleted automatically together with the dimension level. Table 4.2 reveals delete dimension level operation.

| **Delete dimension level** | |
|---|---|
| Syntax with input and output parameters | **Delete_level**(level ***l*** (DL)$\in$ ARR, **I**{$l_{del}$ }) <br> **Input**: Schema DC.$V_i$, instances I.$V_i$, level name $l_{del}$ to be deleted <br> **Output:** new schema DC′.$V_j$, new instances I'$V_j$ |
| Pre-condition(s) | $l_{del} \in$ DL, ARR |
| Post-condition(s) | $l_{del} \notin$ DL, ARR |
| Example | Delete_level(Semester(Time), **I**{$l_{del}$) |
| Semantic expressed by means of the MD schema | **Schema:** <br> S′: **DC** <KA, DA, DL – {$l_{del}$}, ARR′: ARR – ( {$l_{del}$})> <br> **FC**<KA, Gran′: Gran(*f*) – $l_{del}$, FA> |

**Table 4.2: Delete Dimension Level Operation**

### 4.6.2 Modification of a Dimension Attribute

**3. Add Dimension Attribute:** this operation adds a new attribute to a dimension class. Table 4.3 displays Add dimension attribute operation.

| **Add dimension attribute** | |
|---|---|
| Syntax with input and output parameters | **Add_attribute** (new attribute $a_{new}$ (DA), **I**{$DA_{new}$ })<br>**Input**: Schema $DC.V_i$, instances $I.V_i$, attribute $a_{new}$ with dom($a_{new}$) to be added<br>**Output:** new schema $DC'.V_j$, new instances $I'V_j$ |
| Pre-condition(s) | $a_{new} \notin DA$ |
| Post-condition(s) | $a_{new} \in DA$ |
| Example | **Add_attribute** (District(Location), **I** ) |
| Semantic expressed by means of the MD schema | **Schema:**<br>S′: **DC** <KA, DA’: DA $\cup$ {$a_{new}$}, DL, ARR><br>**FC** <KA, Gran, FA> |

**Table 4.3: Add Dimension Attribute Operation**

**4. Delete Dimension Attribute:** this operation deletes an existing dimension attribute from a dimension class. Table 4.4 shows Delete dimension attribute operation.

| **Delete Dimension Attribute** | |
|---|---|
| Syntax with input and output parameters | **Delete_attribute** (del attribute $a_{del}$ (DA), I{$DA_{del}$ })<br>**Input:** Schema $DC.V_i$, instances $I.V_i$, attribute name $a_{del}$ to be deleted<br>**Output:** new schema $DC'.V_j$, new instances $I'V_j$ |
| Pre-condition(s) | $a_{del} \in DA$ |
| Post-condition(s) | $_{adel} \notin DA$ |
| Example | **Delete_attribute** (street (location), I) |
| Semantic expressed by means of the MD schema | **Schema:**<br>S′: **DC** <KA, DA′: DA - {$a_{del}$}, DL, ARR><br>**FC** <KA, Gran, FA> |

**Table 4.4: Delete Dimension Attribute Operation**

### 4.6.3 Modification of a Classification Relationship

**5. Add Classification Relationship:** this operation is triggered by the add dimension level operation. After the add dimension level is performed, the add classification relationship operation then inserts a new relationship between the new added dimension level and existing dimension level in ARR. Figure 4.5 reveals Add classification relationship operation.

| **Add Classification Relationship** | |
|---|---|
| Syntax with input and output parameters | **Add_classification**($l_{new}$ ($l_1$,$l_2$ ),DC, **I** }) <br> **Input**: Schema DC.$V_i$, instances I.$V_i$, new level name $l_{new}$ <br> **Output:** new schema DC′.$V_j$, new instances I'$V_j$ |
| Pre-condition(s) | $l_1 \in L$, $l_2 \in L$, $\{( l_{new} \notin ARR)\}$ |
| Post-condition(s) | $l_{new} \in ARR$ |
| Example | **Add_classification**(semester, (month,year), **Time**, **I**) |
| Semantic expressed by means of the MD schema | **Schema:** <br> S′= **DC** <KA, DA, DL $\cup$ {$l_{new}$},ARR′ =ARR $\cup$\{($l_1$, $l_{new}$ ) $\cup$ ($l_{new}$ $l_2$)\}> <br> **FC** <KA, Gran′: Gran(*f*) $\cup$ $l_{new}$ , FA> |

**Table 4.5: Add Classification Relationship Operation**

**6. Delete Classification Relationship:** this operation deletes an existing relationship between dimension level. The operation is followed by the delete dimension level operation. Figure 4.6 displays Delete classification relationship operation.

| **Delete Classification Relationship** | |
|---|---|
| Syntax with input and output parameters | **Delete_classification**($l_{del}$ ($l_1$,$l_2$ ),DC, **I** }) <br> **Input**: Schema DC.$V_i$, instances I.$V_i$, level name $l_{del}$ to be deleted <br> **Output:** new schema DC′.$V_j$, new instances I’$V_j$ |
| Pre-condition(s) | $l_1 \in L$, $l_2 \in L$, $l_{del} \in ARR$ |
| Post-condition(s) | $l_{del} \notin ARR$ |
| Example | **Delete_classification**(semester, (month,year), **Time**, **I**) |
| Semantic expressed by means of the MD schema | **Schema:** <br> S′: DC <KA, DA, DL - {$l_{del}$},ARR′ =ARR - {($l_1$, $l_{del}$ ) $\cup$ ($l_{del}$ ,$l_2$)}> <br> FC <KA, Gran′: Gran(*f*) – $l_{del}$ , FA, > |

**Table 4.6: Delete Classification Relationship Operation**

### 4.6.4 Modification of a Fact

**7. Add Dimension to Fact**: this operation adds a new dimension class (specified by a dimension level) to an existing fact, thus increasing the number of fact domains. Table 4.7 shows an operation of Add dimension to fact class

| **Add dimension level to fact** | |
|---|---|
| Syntax with input and output parameters | **Add_dimension_into_fact**((DC),L, FC, **I**{f})<br>**Input**: Schema FC.$V_i$, instances I.$V_i$, DC($l_{new}$)<br>**Output:** new schema FC′.$V_j$, new instances I′$V_{\mathbf{j}}$ |
| Pre-condition(s) | L(DC) $\notin$ FC |
| Post-condition(s) | L(DC) $\in$ FC |
| Example | **Add_dimension_into_fact**(Customer(Customer_ID, **I**(f)) |
| Semantic expressed by means of the MD schema | **Schema:**<br>S′: **DC** <KA, DA, DL, ARR><br>**FC** <KA, Gran′: Gran(*f*) $\cup$ {DC($l_{new}$)}, FA> |

**Table 4.7: Add Dimension to Fact Class Operation**

**8. Delete Dimension from Fact:** this operation deletes an existing dimension class, specified by dimension key attribute, from a fact. The instances of fact and dimension have been automatically deleted. Thus, the fact domain needs to be recomputed. Table 4.8 reveals an operation of Delete dimension from fact class.

| **Delete Dimension from fact** | |
|---|---|
| Syntax with input and output parameters | **Delete_dimension_from_fact**((DC),L, FC, **I**{f})<br>**Input**: Schema FC.$V_i$, instances I.$V_i$, DC($l_{del}$)<br>**Output:** new schema FC′.$V_j$, new instances I′$V_j$ |
| Pre-condition(s) | DC $\in$ FC |
| Post-condition(s) | DC $\notin$ FC |
| Example | **Delete_dimension_from_fact** (Customer(Customer_ID), Sales, **I**(f)) |
| Semantic expressed by means of the MD schema | **Schema:**<br>S′: DC <KA, DA, ARR><br>FC <KA, Gran′: Gran - {DC($l_{del}$)}, FA> |

**Table 4.8: Delete Dimension from Fact Class Operation**

**9. Add Fact Attribute:** this operation adds a new fact attribute or measure to a fact class. Table 4.9 displays Add fact attribute operation.

| **Add fact attribute** | |
|---|---|
| Syntax with input and output parameters | **Add_FA** (new attribute $a_{new}$ (FC), $\mathbf{I}_s$)<br>**Input**: Schema FC.$V_i$, instances I.$V_i$ , new attribute $FA_{new}$ with dom($FA_{new}$) to be added<br>**Output:** new schema FC′.$V_j$ , new instances I'$V_j$ |
| Pre-condition(s) | $FA_{new} \notin FC$ |
| Post-condition(s) | $FA_{new} \in FC$ |
| Example | **Add_FA** (avg_sales(Sales)) |
| Semantic expressed by means of the MD schema | **Schema:**<br>S′: **DC** <KA, DA, DL, ARR><br>**FC** <KA, Gran, FA′: FA $\cup$ {$FA_{new}$}> |

**Table 4.9: Add Fact Attribute Operation**

**10. Delete Fact Attribute:** this operation deletes the existing fact attribute from a fact class. Table 4.10 shows Delete fact attribute operation.

| **Delete fact attribute** | |
|---|---|
| Syntax with input and output parameters | **Delete_FA** (del attribute $a_{del}$ (FC), $I_s$)<br>**Input:** Schema FC.$V_i$, instances I.$V_i$, attribute name $FA_{del}$ to be deleted<br>**Output:** new schema FC′.$V_j$, new instances I'$V_j$ |
| Pre-condition(s) | $FA_{del} \in FC$ |
| Post-condition(s) | $FA_{del} \notin FC$ |
| Example | **Delete_FA** (dollar_costs(Sales), I) |
| Semantic expressed by means of the MD schema | **Schema:**<br>S′: **DC** <KA, DA, DL,ARR><br>**FC** <KA, Gran, FA′: FA - {$FA_{del}$}> |

**Table 4.10: Delete Fact Attribute Operation**

## 4.7 A Summarized Conceptual Approach of Schema Evolution in OOMDB: Structural and Instance Propagation

To demonstrate the overall process of the integration idea of OOMDB schema evolution, we present its synopsis idea in Figure 4.7

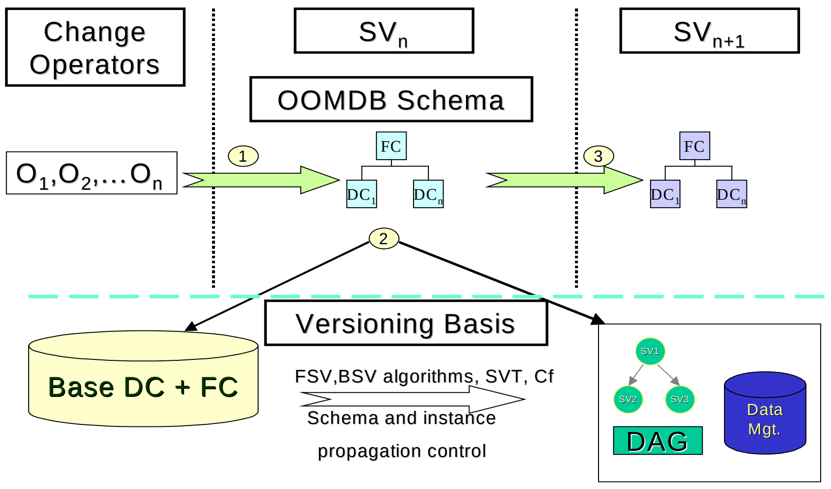


**Figure 4.7: Handling Schema Evolution in OOMDB: Structural and Instance Propagation**

Basically, the change of OOMDB schema is initiated from the evolution operations. Then the process of evolution control is started. Here, our OOMDB schema versioning approach plays a major role in conducting the reliable process of schema changes as well as the schema version control. In this step, the changes of the schema and instance are well controlled by FSV algorithm, BSV algorithm, SVT function and *Cf* function. Also, the structural schema versions are maintained by the schema version storage-- DAG and their respective instances are maintained by their base classes and data management as well. After the versioning mechanism is completely performed, the next version of schema is generated.

## 4.8 MD Schema Version Consistency

To check the consistency of database schema, there is a need to ensure that after the modifications, the structural schema is transformed into the consistent state and its instances are properly propagated to the modified schema.

In our approach, we ensure the consistency in both the schema and instance level by the *SVT* and *Cf* defined during the derivation path in DAG. The proof of Lemma 4.5.1 shows the existence and uniqueness of schema derivation properties. Therefore, it is ensured that the schema version transformation and instance propagation always exist and yield the valid solution.

# CHAPTER 5

# DESIGN AND IMPLEMENTATION

## 5.1 Preliminary

This chapter describes the design and implementation of the **OOMDB Schema Evolution Manager** prototype system in terms of object class design, database design, system architecture design, and data flow diagram. The prototype system demonstrates the idea of the proposed schema evolution framework in supporting the change of multidimensional database schema in both the schema adaptation and instance propagation. The schema evolution manager is responsible for schema evolution support and versions control. The prototype is designed to support schema changes by the proposed evolution operations, and the system will keep all changes of schema in version storage. In order to validate the consistency and the correctness of the query results of historical versions, many query scenarios for several schema versions were thus performed. And its test results were validated with OLAP Tools MS. SQL Analysis Manager. The experimental design and the evaluation are described in the next chapter.

## 5.2 Object Class Design

In this section, we describe the design of the object classes used for implementation. Figure 5.1 shows the relationship of the object classes existing in the system.

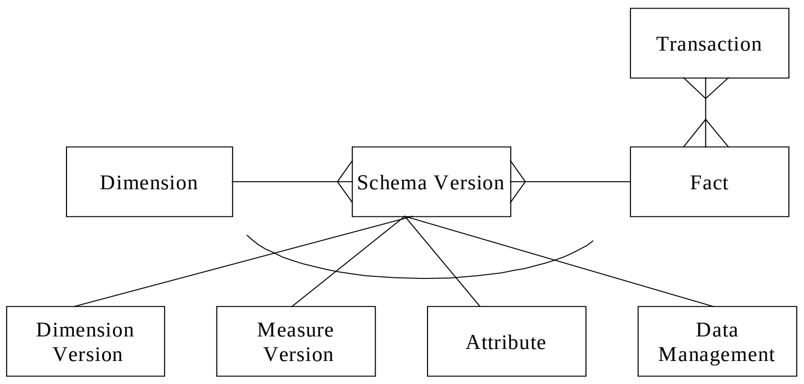


**Figure 5.1: Relationship between Objects**

There are eight major object classes designed for the OOMDB SEMAN implementation. The object classes include dimension, fact, transaction, schema version, dimension version, measure version, attribute, and data management. The latter four are the subclass of schema version object class. Since these object classes are designed corresponding to the meta relational table, the definition of these object classes is associated with the definition of the respective meta table discussed in the section of database design.

Below, all object classes are defined with respect to their definition, properties and methods.

1. **Class Dimension**

Definition: Class that stores the dimension base schema and object instances.

Properties

Dimension ID (DID)

Attribute_Name (DA)

Level_Name(DL)

Attribute_roll up_relationship (ARR)

Methods

Add Dimension Attribute (Add_DA)

Delete Dimension Attribute (Del_DA)

Add Dimension Level (Add_DL)

Delete Dimension Level (Del_DL)

Add Dimension Class (Add_DC)

Delete Dimension Class (Del_DC)

2. **Class Fact**

Definition: Class that holds the fact base schema and fact instances.

Properties

<u>Fact_ID (FID)</u>

Gran (DID)

Attribute_Name (FA)

Methods

Add Fact Attribute (Add_FA)

Delete Fact Attribute (Del_FA)

Define Measure (Define)

Update Fact properties (Update_Property)

Data slicing

Data dicing

Rolling up

Drilling down

Sum

Average

Max

Min

Count

### 3. Class Transaction

Definition: Class that contains the transactional data. This class incorporates the numeric value of the measure and the given dimension that will be further used in creating fact class and cube.

Properties

Transaction ID (TID)
Dimension_ID (DID)
Measure(M)

Methods

Add Transaction
Update Transaction
Delete Transaction

### 4. Class Schema Version

Definition: Class that houses the schema change records of all schema version. This class and its subclass are exploited to support the main feature of the prototype OOMDB SEMAN that controls the semantic of change and change propagation. It is a super class, which has four subclasses: dimension version, measure version, attribute, and object version.

Properties

Schema Version_ID (SV_ID)
TimeStart_TimeEnd (TS_TE)
Change_Operator (O)
Modified_Class (MC)

Methods

Add change record
Assign schema version time
Find schema version properties
Selection

Projection

Sort record

Join table

**5. Class Dimension Version**

Definition: Class that bears the information or record of dimension schema for all schema versions.

Properties

Schema Version_ID (SV_ID)

No_of_Dimension (No_D)

Dimension_Name (DC)

Methods

Add change record

Find dimension version properties

**6. Class Measure Version**

Definition: Class that stores the information or record of fact schema for all schema versions

Properties

Schema Version_ID (SV_ID)

No_of_Measure (No_FA)

Measure_Name (FA)

Function (F)

Methods

Add change record

Find measure version properties

**7. Class Attribute**

Definition: Class that contains the information or record of the structure of dimension attribute and dimension level of the dimension class for all schema versions

Properties

Schema Version_ID (SV_ID)

No_of_Dimension (No_ D)

No_of_Attribute(No_ A)

Attribute_Name (AName)

Attribute_Type (Type)

Methods

Add change record

Find Attribute properties

**8. Class Data Management**

Definition: Class that contains the records of the object instance manipulation by means of the SQL statement. This class is helpful for tracing and retrieving the legal instance when the change has taken place in the instance level.

Properties:

Schema Version_ID (SV_ID)

Command_Line (Command)

No_SQL (No_SQL)

Methods:

Add command line

Update command line

Delete command line

View command line

## 5.3 Database Design

All instances of the object classes residing in the meta relational table have the same name as their classes. For example, object class Dimension is associated with “Dimension Table”. The relational schema and table structure of the meta table are listed as follows:

### Relational Schema

**Dimension** (DID: integer, DA: string, DL: string, ARR: string)
**Fact** (FID: integer, DID: integer, FA: string)
**Transaction** (TID: integer, DID:integer, FA:string)
**Schema Version** (SV_ID:integer, TS_TE: date/time, O: string, MC:string)
**Dimension Version** (SV_ID:integer, No_D:integer, DC: string)
**Measure Version** (SV_ID:integer, No_FA:integer, FA: string, F:string)
**Attribute** (SV_ID:integer, No_D:integer, No_A:integer, A_name:string, Type:Boolean)
**Data Management** (SV_ID:integer, Command_Line:string, no_SQL:integer)

### Table Structure

Table 5.1: Table Structure of Dimension

| Attribute Name | Attribute Type | Attribute Size | Full Description |
|---|---|---|---|
| DID | Number | 4 | Dimension ID |
| DA | Text | 50 | Dimension Attribute Name |
| DL | Text | 50 | Dimension Level Name |
| ARR | Text | 50 | Attribute roll-up relationship description |

Table 5.2: Table Structure of Fact

| Attribute Name | Attribute Type | Attribute Size | Full Description |
|---|---|---|---|
| FID | Number | 4 | Fact ID |
| DID | Number | 4 | Dimension ID (dimension class associated with the FC) |
| FA | Text | 30 | Fact attribute name |

Table 5.3: Table Structure of Transaction

| Attribute Name | Attribute Type | Attribute Size | Full Description |
|---|---|---|---|
| TID | Number | 4 | Transaction ID |
| DID | Number | 4 | Dimension ID |
| FA | Text | 30 | Fact attribute name |

Table 5.4: Table Structure of Schema Version

| Attribute Name | Attribute Type | Attribute Size | Full Description |
|---|---|---|---|
| SV_ID | Number | 4 | Schema Version No. |
| TS_TE | Date/Time | 20 | Time Start and Time End of the schema version |
| O | Text | 30 | Schema evolution operations |
| MC | Text | 50 | DC or FC Class name which is modified |

Table 5.5: Table Structure of Dimension Version

| Attribute Name | Attribute Type | Attribute Size | Full Description |
|---|---|---|---|
| SV_ID | Number | 4 | Schema Version No. |
| No_D | Number | 20 | The number of dimension of each schema version |
| DC | Text | 30 | List of Dimension Name of the schema version |

Table 5.6: Table Structure of Table Measure Version

| Attribute Name | Attribute Type | Attribute Size | Full Description |
|---|---|---|---|
| SV_ID | Number | 4 | Schema Version No. |
| No_FA | Number | 20 | The number of fact attributes of each schema version |
| FA | Text | 30 | List of Fact attribute Name of the schema version |
| F | Text | 30 | OLAP functions |

Table 5.7: Table Structure of Attribute

| Attribute Name | Attribute Type | Attribute Size | Full Description |
|---|---|---|---|
| SV_ID | Number | 4 | Schema Version No. |
| No_D | Number | 20 | The number of dimensions of each schema version |
| No_A | Number | 20 | The number of attributes of each dimension class for each schema version |
| A_Name | Text | 30 | Attribute Name |
| Type | Boolean | 2 | DA or DL |

Table 5.8: Table Structure of Data Management

| Attribute Name | Attribute Type | Attribute Size | Full Description |
|---|---|---|---|
| SV_ID | Number | 4 | Schema Version No. |
| Command_Line | Text | 80 | SQL command_line |
| NO_SQL | Number | 4 | Sequence of SQL command |

## 5.4 OOMDB SEMAN System Architecture Design

The system architecture and functional modules of OOMDB SEMAN are based on OOMDB model together with the object-oriented versioning approach. The purpose of the prototype is to demonstrate that the proposed schema evolution framework is feasible in supporting the change of multidimensional database schema in both the schema adaptation and instance propagation.

The following figure shows the functional properties of OOMDB SEMAN.

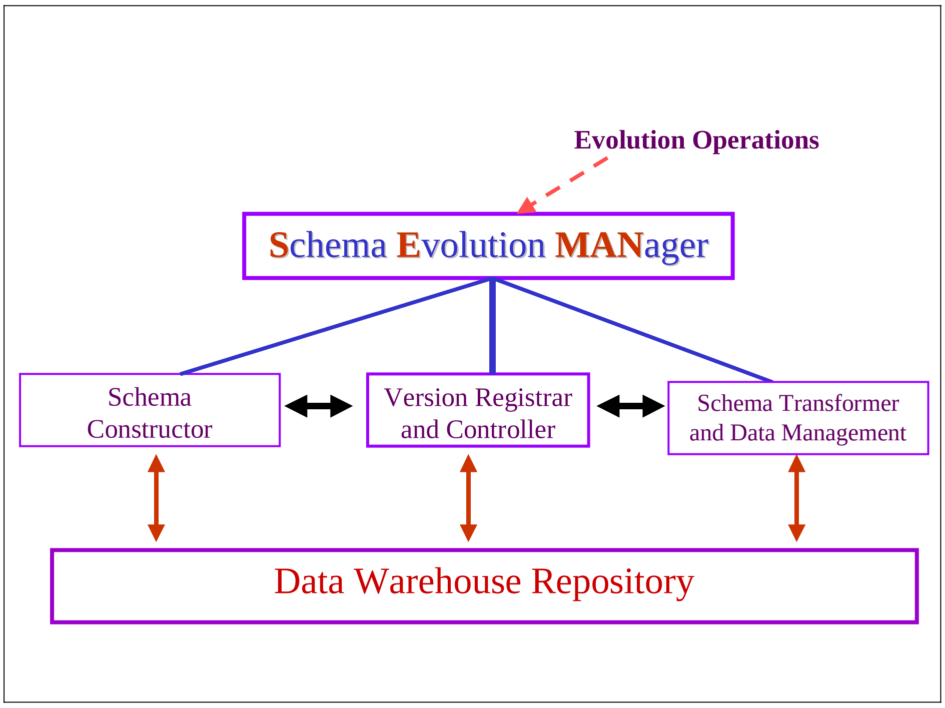

Evolution Operations
Schema Evolution MANager
Schema Constructor
Version Registrar and Controller
Schema Transformer and Data Management
Data Warehouse Repository

**Figure 5.2: System Architecture**

The changes of schema are controlled by Version Registrar and Controller. Since the change may take place at either schema or instance level, we thus introduce the Schema Transformer and Instance Adapter which help maintain the correctness and consistency in both levels.

**Schema Constructor:** The schema constructor serves for building a conceptual OOMDB schema with respect to the structural schema version. From the schema constructor, the designer can perform schema modification of any elements of the multidimensional database schema we have proposed in chapter 4.

**Schema Transformer and Data Management:** This function is an important component in controlling the change of multidimensional database schema. The engine is to force the structural schema change from one schema version to another newer one. Actually, this system module is totally influenced by Schema Version Transform Function (SVT). Also, the Instance Adapter is designed to work with the Schema Transformer and to perform its role in controlling the change in the instance level. Whenever the multidimensional database schema has been changed, which affects the instance of the promising classes, this system part will propagate the legal instance to the proper status of the particular schema version. This system module is based on the management of conversion function **(*Cf*).**

**Version Registrar and Controller:** The functions of the component are recording, controlling, and retrieving the versions of database schema. The two former components collaboratively work to this functional module. The Forward Schema Version Compatibility Algorithm and Backward Schema Version Compatibility Algorithm play a major role in conducting evolution tasks, and support the overall system functions in an efficient manner.

In point of DW/OLAP schema designer's view, OOMDB SEMAN offers an easy-to-use function to perform the schema update and allow OLAP designer to retrieve schema objects anytime in their evolution histories. In addition, we develop the multidimensional query mechanism to support the queries made on the different schema version.

## 5.5 The Data Flow Diagram

In this section, we present the system experiment process with the data flow diagram. Figure 5.3 illustrates the context diagram of the experimental system.

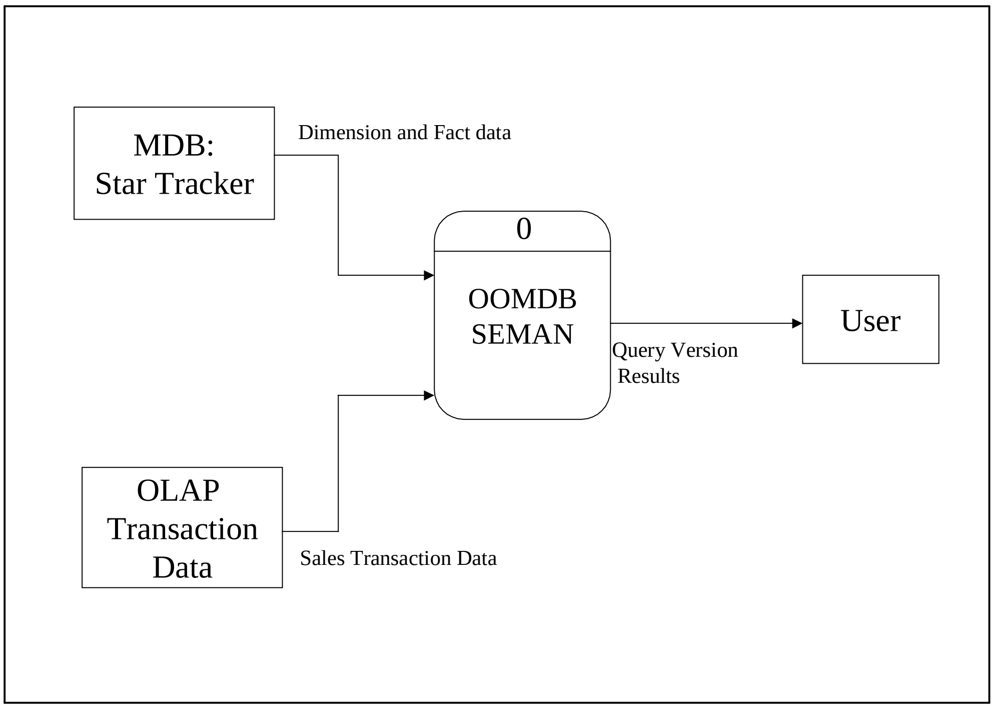


**Figure 5.3: Context Diagram of the Experimental System**

### 5.5.1 Two Major Processes of the Experimental System

The prototype system consists of two major processes described as follows:

**Process 1: Create Schema Evolution Manager**

This process creates the schema evolution manager, which is an important component in constructing OOMDB schema, controlling structural schema version, and adapting dimension and instance version. Basically, the inputs are obtained from the base dimension and fact table while the output of the process is an evolved schema version, and each of the profiles is written to the schema version table file.

**Process 2: Create Query Processing**

This process creates the query processing module. Inside the process, the schema version stored in the schema version table is retrieved to construct a specified cube version coupled with the dimension and fact data (the fact data also comes from the transaction data). As a consequence, the output of the process is the query result of the specified schema version.

The data flow diagram of the two major processes is illustrated in Figure 5.4

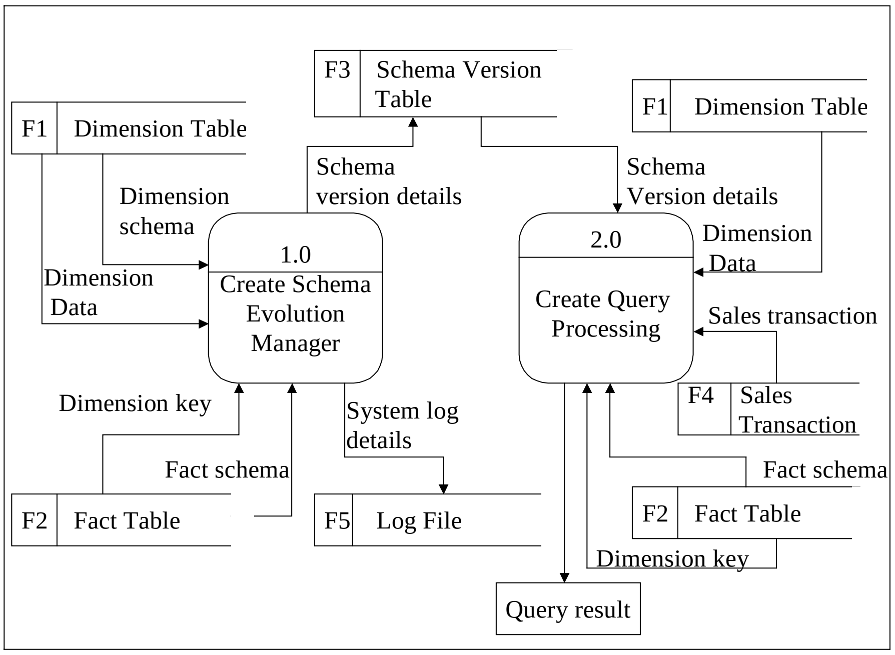


**Figure 5.4: DFD Level 1 of the Experimental System**

Figure 5.5 and 5.6 detail the sub-process of the two afore-mentioned major processes in more detailed DFD level.

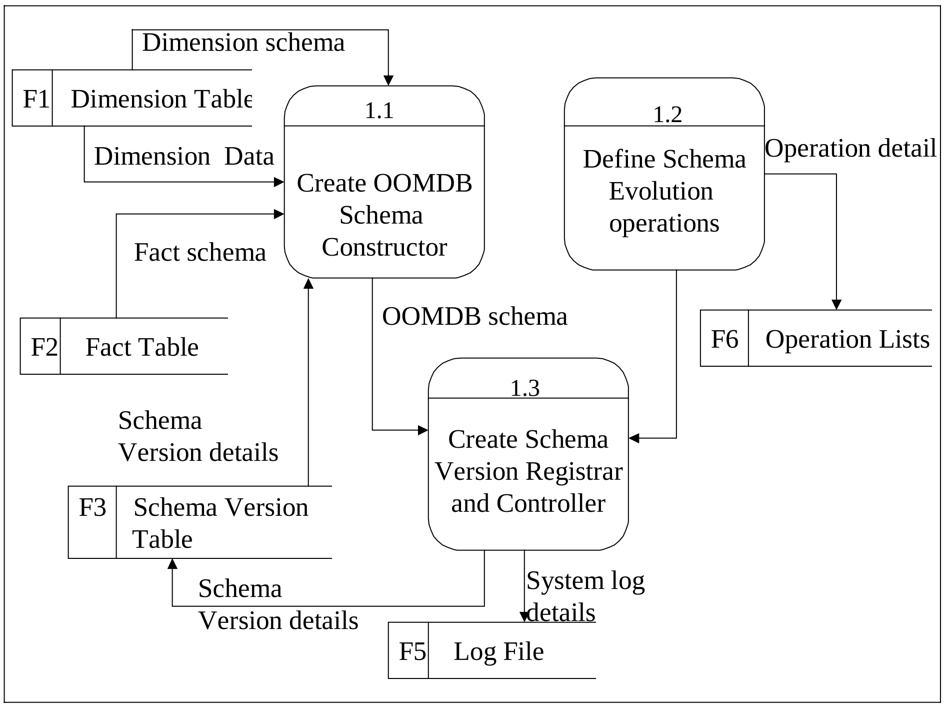


**Figure 5.5: DFD: Create Schema Evolution Manager**

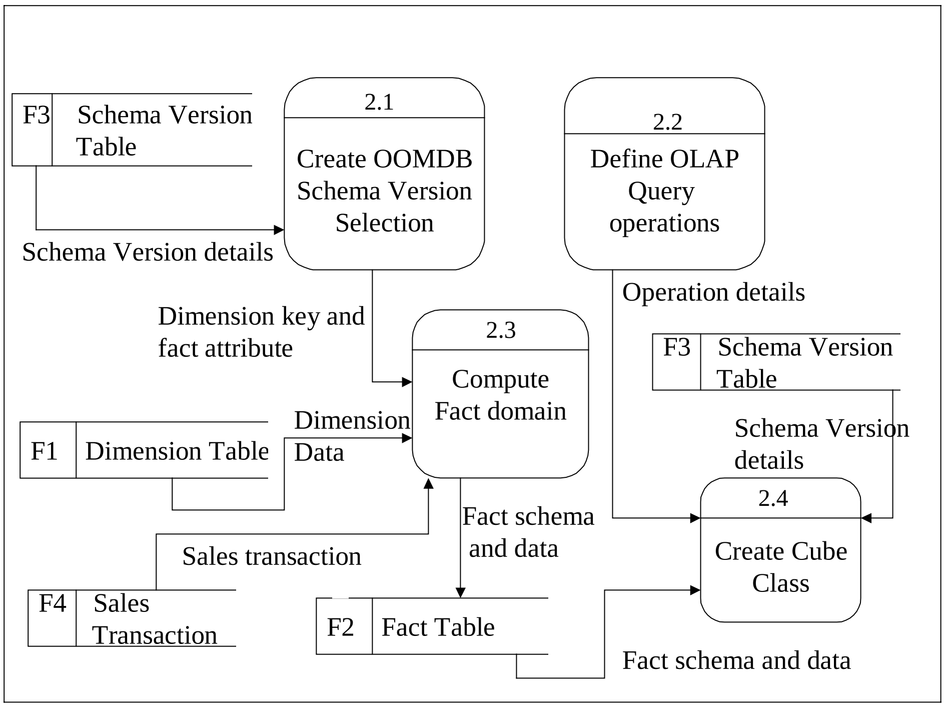


**Figure 5.6: DFD: Create Query Processing**

Further details of its functions are described in the experimental framework in the next chapter.

# CHAPTER 6

# EXPERIMENT AND RESULT

## 6.1 Scope of the Experiment

The experiment is conducted to verify the idea of a novel approach of the multidimensional database schema evolution framework proposed in chapter 4. The chapter begins with a multidimensional framework of the experimental system. The experiment and result are shown in later sections.

## 6.2 Experimental Framework

The experiment process deals with the development of prototype features that we have detailed in system architecture design. The system consists of five major functional modules: OOMDB Schema Version Constructor and Controller, OLAP Transaction Data Generator, Data Management Function, System Log and Query Processing. We then describe these features with the screen design.

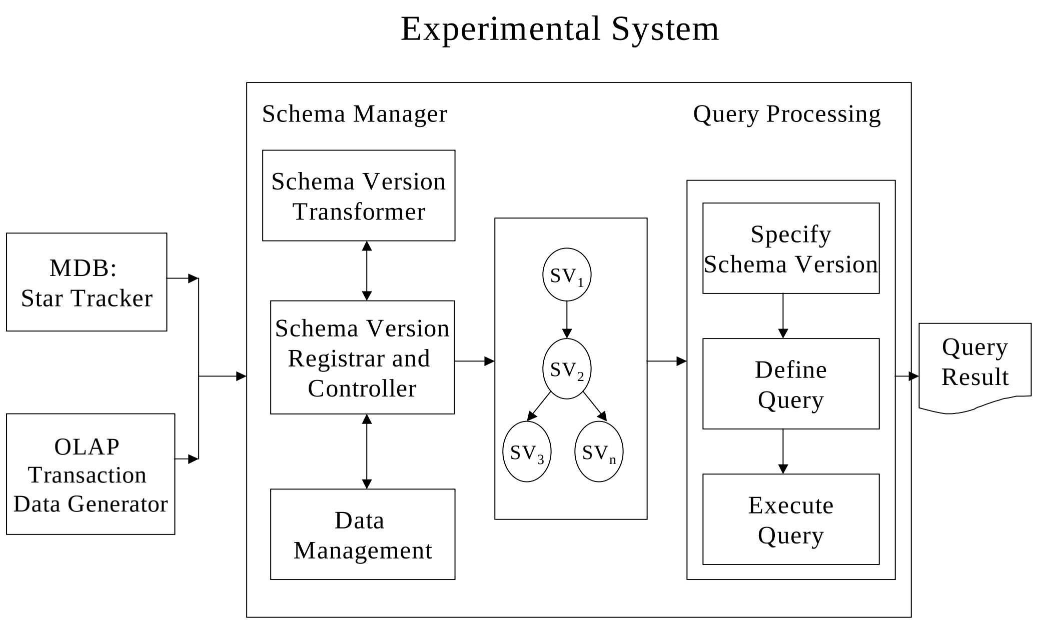


**Figure 6.1: Experimental Framework**

### 6.2.1 Experimental Data

In the experiment, we use two types of data for the implementation: Benchmark data taken from Star Tracker$^{TM}$ [29] and OLAP transaction data.

➢ **Benchmark Data**

The MD benchmark data and the database schema used for multidimensional database schema evolution analysis is derived from the grocery store system developed by Ralph Kimball [29]. The base schema incorporating fact and dimension is thus primarily based on the grocery store system. Below is the framework of a grocery multidimensional database used in our experiment.

**Dimension:**

- Product
- Promotion
- Store
- Time
- Supplier
- Customer
- Sales_person

**Fact Table:**

- Sales Fact

➢ **OLAP Transaction Data**

We also develop an OLAP transaction data generator to test a high volume of data. Table 6.1 summarizes volume of transaction data and the necessary statistics used in the experiment.

**Table 6.1 Sample Size of Experimental Data**

| Schema Version | Test Queries | OLAP Transaction Data |
|---|---|---|
| SV1 | 50 | 1,002,830 |
| SV2 | 50 | 1,014,270 |
| SV3 | 50 | 1,110,250 |
| . . . | . . . | . . . |
| SV100 | 50 | 1,208,760 |
| Total schema version : 100 SVs. All combination cases: 100 * 50= 5,000 with up to1,000,000 records of transaction data for each schema version | | |

Thoroughly, all schema versions are tested with the sequences of evolution operations we applied case by case. In each case, the basic schema evolution operations such as Add DA, Del DA, Add DL, Del DL,…, Del FA are distributively applied to the test cases. Therefore, our test plan actually covers all potential events of schema changes that we have already defined in chapter 4. Significantly, the test scheme is achieved with the 5,000 combination cases of which the 50 queries are applied for each schema version with the substantial size of transaction data.

### 6.2.2 File Structure of Grocery Multidimensional Database

Below is the file structure of the system mapped from the object-oriented multidimensional database schema. Some attributes are modified in order to fit the system purpose. The experimental grocery store database is developed in MS SQL Server 2000, whereas its object classes, codes and user interfaces are built in Visual Basic and Power Builder. File Structures of all these tables are described as follows:

Table 6.2: File Structure of Dimension Table “Product”

| Field Name | Data Type | Data Size | Description |
|---|---|---|---|
| Product_key | Number | Long Int | Product Object ID(KA) |
| Product_name | Text | 50 | Product description |
| Full_description | Text | 255 | Product full description |
| SKU_number | Number | Byte | SKU number |
| Brand | Text | 255 | Product brand |
| Subcategory | Text | 255 | Sub category of product |
| Category | Text | 255 | Category of product |
| Department | Text | 255 | Department of product |
| Supplier_key | Number | Long Int | Supplier Object ID (FK) |

Table 6.3: File Structure of Dimension Table “Promotion”

| Field Name | Data Type | Data Size | Description |
|---|---|---|---|
| Promotion_key | Number | Long Int | Promotion Object ID(KA) |
| Promotion_name | Text | 50 | Promotion description |
| Price_reduction_type | Text | 255 | Price reduction type |
| Ad_type | Text | 255 | Advertising type |
| Display_type | Text | 255 | Display type |
| Coupon_type | Text | 255 | Coupon type |
| Ad_media_type | Text | 255 | Advertising media type |
| Display_provider | Text | 255 | Display provider |

Table 6.4: File Structure of Dimension Table “Store”

| Field Name | Data Type | Data Size | Description |
|---|---|---|---|
| Store_key | Number | Long Int | Store object ID (KA) |
| Store_name | Text | 50 | Store description |
| City | Text | 255 | City in which a store located |
| Store_country | Text | 255 | Country in which a store located |
| Store_state | Text | 255 | State in which a store located |
| Sales_region | Text | 255 | Region in which a store located |
| Floor_plan_type | Text | 255 | Floor plan type |
| Photo_proc_type | Text | 255 | Photo Processing type |
| Finance_serv_type | Text | 255 | Finance service type |

Table 6.5: File Structure of Dimension Table “Dtime”

| Field Name | Data Type | Data Size | Description |
|---|---|---|---|
| Dtime_key | Number | Long Int | Dtime Object ID(KA) |
| Ddate | Date/Time | 8 | Date |
| Day_of_week | Text | 255 | Day of week |
| Day_number_in_month | Number | Byte | Number of day in a month |
| Week_number_in_year | Number | Byte | Number of week in a year |
| Month | Text | 50 | Month |
| Quarter | Number | Byte | Quarter |
| Fiscal_period | Text | 255 | Fiscal period |
| Year | Number | Byte | Year |
| Holiday_flag | Text | 255 | Holiday flag |

In addition to the above base dimension tables, the three more dimension tables(dimension classes): “Supplier,” “Customer” and “Sales_person” are generated to substantiate the object evolution purpose (see Tables 6.6, 6.7 and 6.8, respectively). Finally, the file structure of fact table “Sales Fact” is revealed in Table 6.9.

Table 6.6: File Structure of Dimension Table “Supplier”

| Field Name | Data Type | Data Size | Description |
|---|---|---|---|
| Supplier_key | Number | Long Int | Supplier Object ID(KA) |
| Supplier_name | Text | 50 | Supplier Name |
| Supplier_city | Text | 255 | City in which a store located |
| Supplier_state | Text | 255 | State in which a store located |
| Supplier_zip | Number | 10 | Supplier zip code |

Table 6.7: File Structure of Table “Customer”

| Field Name | Data Type | Data Size | Description |
|---|---|---|---|
| Customer_key | Number | Long Int | Customer Object ID(KA) |
| Company_name | Text | 50 | Customer company name |
| Contact_name | Text | 50 | Name of contact person |
| Contact_title | Text | 50 | Position of contact person |
| City | Text | 255 | City in which a customer lives |
| Store_country | Text | 255 | Country in which a customer lives |
| Store_state | Text | 255 | State in which a customer lives |
| Phone | Text | 50 | Customer tel no. |

Table 6.8: File Structure of Table “Sales_person”

| Field Name | Data Type | Data Size | Description |
|---|---|---|---|
| Person_key | Number | Long Int | Sales Person Object ID(KA) |
| First_name | Text | 50 | Sales Person first name |
| Last_name | Text | 50 | Sales Person last name |
| City | Text | 255 | City in which a sales person works |
| Store_country | Text | 255 | Country in which a sales person works |
| Store_rigon | Text | 255 | State in which a sales person works |
| Home_phone | Text | 50 | Sales person tel no. |

Table 6.9: File Structure of Fact Table “Sales Fact”

| Field Name | Data Type | Data Size | Description |
|---|---|---|---|
| Sales_fact_key | Number | Long Int | Sales fact Object ID(KA) |
| Product_name | Text | 50 | Promotion description |
| Promotion_name | Text | 50 | Promotion description |
| Store_name | Text | 50 | Store Description |
| Date | Date/Time | 8 | Date |
| Dollar_sales | Number | Byte | Dollar sales |
| Unit_sales | Number | Byte | Unit Sales |
| Dollar_cost | Number | Byte | Dollar cost |
| Customer_count | Number | Byte | Number of customers |

### 6.2.2 OOMDB SEMAN System Functions

- **OOMDB Schema Version Constructor and Controller**

The schema version constructor and controller screen is the major system function of the OOMDB SEMAN. Its function is managed by the object-oriented versioning concept. Therefore, schema evolution is formally treated by this functional module. Figure 6.2 presents the basic construct of the OOMDB schema consisting of dimension classes and fact class. The structural schema is shown in terms of tree structure. Formally, each prime node of tree represents the class of dimension and fact. Here, fact attributes, dimension attributes and dimension levels reside in the respective class in a sequence of schema version. This screen presents the structural schema in any specified version. OLAP designer can go to any structural schema version by selecting the version identifier in drop down list. To manipulate or change the database schema, database administrator or designer can modify the schema by means of the schema evolution operations at the node of the MD schema tree.

In addition, the right hand side of the main screen is a graphical conceptual schema representation corresponding to the specified database schema version. The screen shot is illustrated in Figure 6.2.

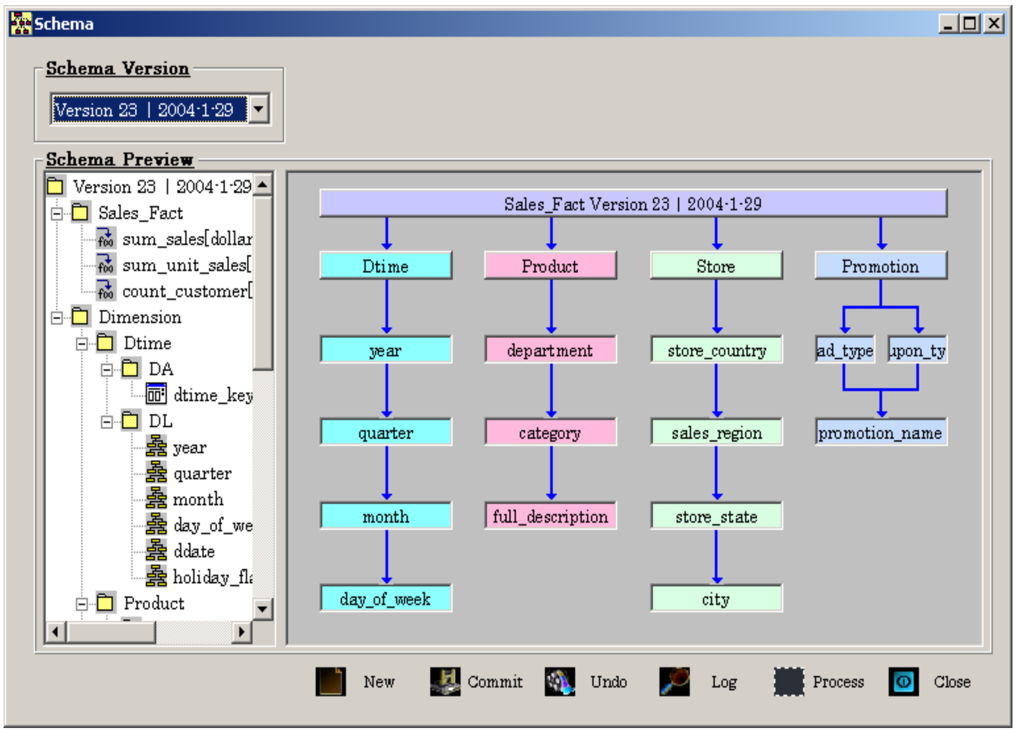


**Figure 6.2: Schema Version Constructor and Controller**

In addition to the base data including the dimension data and fact data of the real world grocery database, we also develop an OLAP transaction data generator to test a high volume of data (see Figure 6.3).

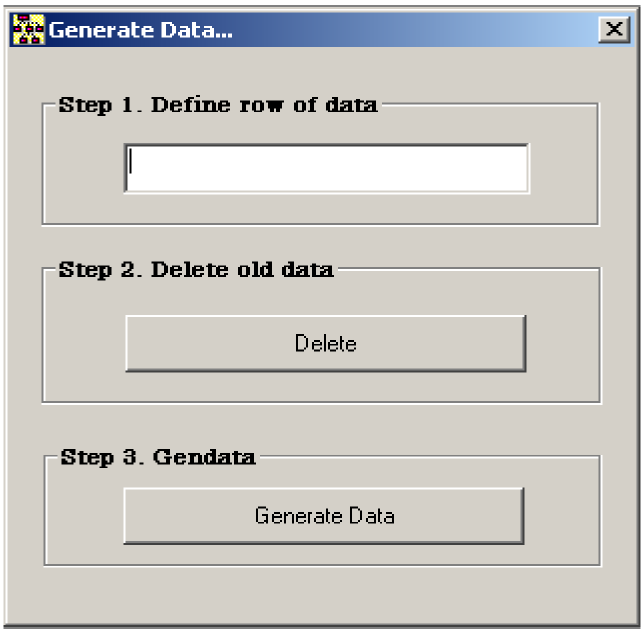
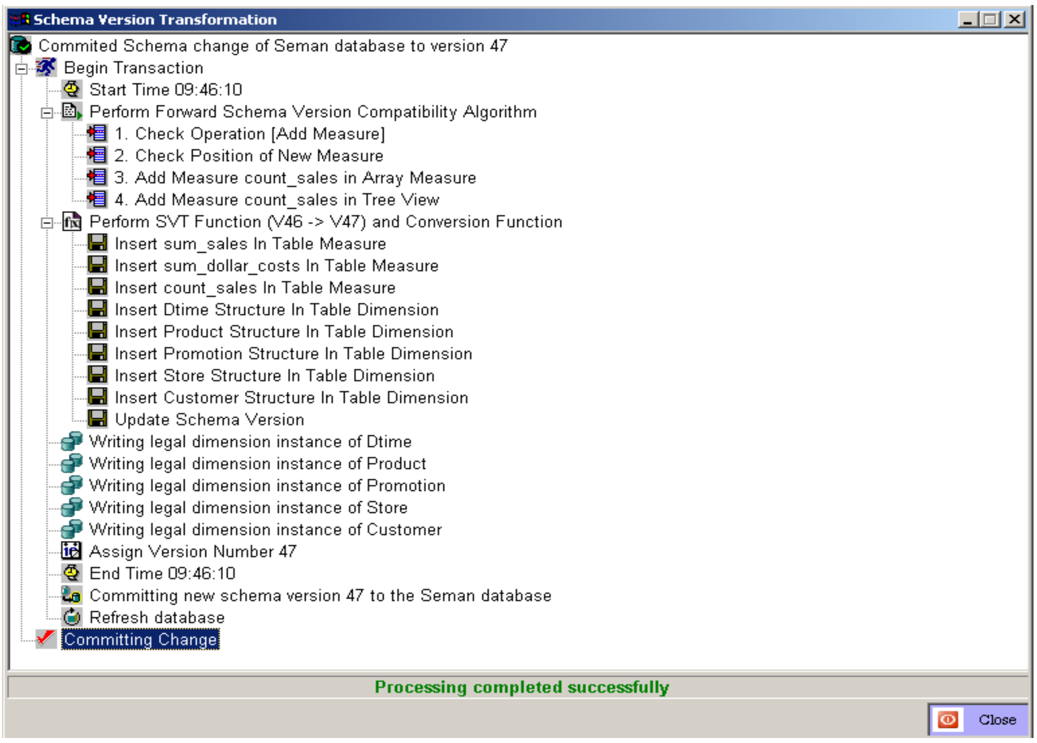


**Figure 6.3: OLAP Transaction Data Generator**

The idea of this function is that the user can enter the no. of data record that he wants to test. Then the system will generate the transaction data as request.

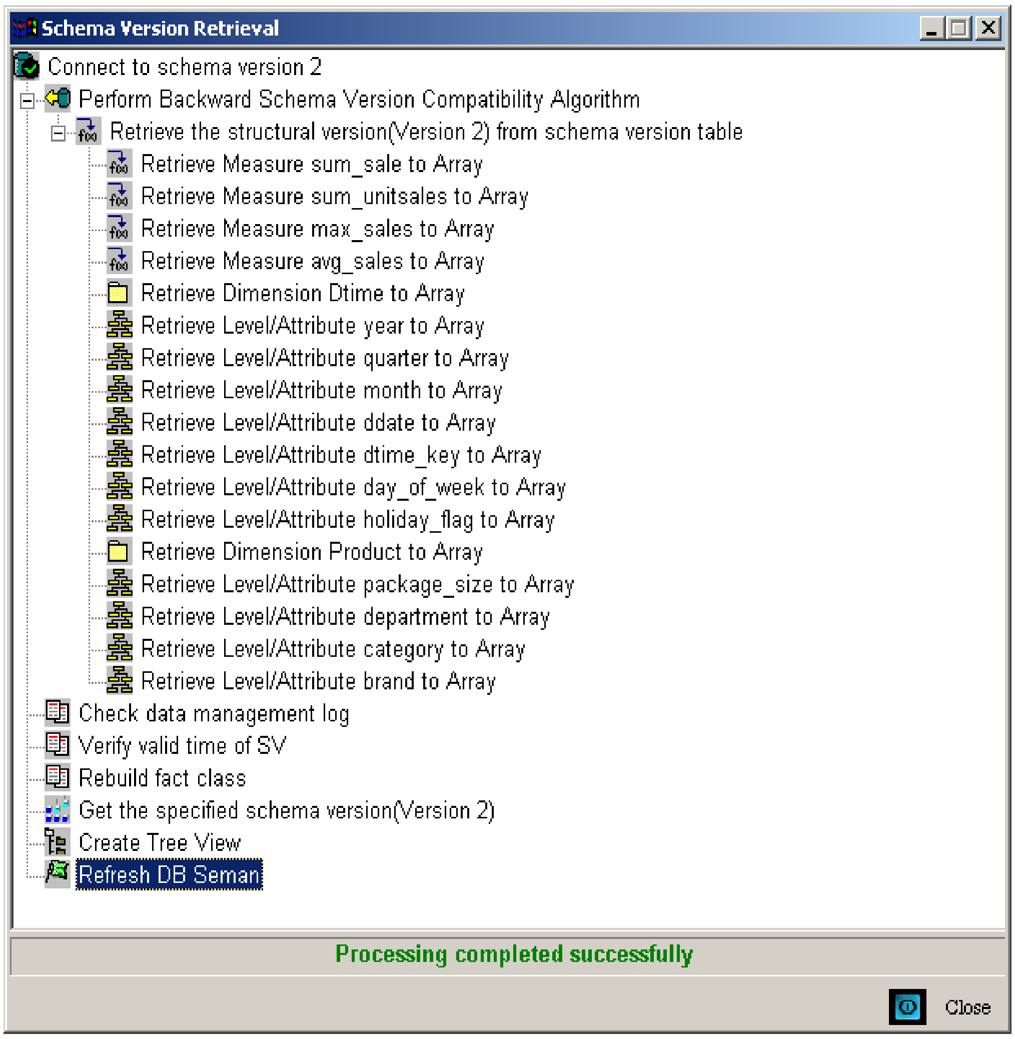
To trace the evolution process, the system shows the steps of schema change that is controlled by the relevant algorithms. Figure 6.4 displays the screen shot of process of detailed changes.



```
Schema Version Transformation
Commited Schema change of Seman database to version 47
Begin Transaction
    Start Time 09:46:10
    Perform Forward Schema Version Compatibility Algorithm
        1. Check Operation [Add Measure]
        2. Check Position of New Measure
        3. Add Measure count_sales in Array Measure
        4. Add Measure count_sales in Tree View
    Perform SVT Function (V46 -> V47) and Conversion Function
        Insert sum_sales In Table Measure
        Insert sum_dollar_costs In Table Measure
        Insert count_sales In Table Measure
        Insert Dtime Structure In Table Dimension
        Insert Product Structure In Table Dimension
        Insert Promotion Structure In Table Dimension
        Insert Store Structure In Table Dimension
        Insert Customer Structure In Table Dimension
        Update Schema Version
    Writing legal dimension instance of Dtime
    Writing legal dimension instance of Product
    Writing legal dimension instance of Promotion
    Writing legal dimension instance of Store
    Writing legal dimension instance of Customer
    Assign Version Number 47
    End Time 09:46:10
    Committing new schema version 47 to the Seman database
    Refresh database
Committing Change
Processing completed successfully
Close
```

**Figure 6.4: Schema Version Transformation Details**

Similarly, the system also offers the schema retrieval details by means of the backward schema version compatibility algorithm. Figure 6.5 reveals the screen shot of a process of schema version retrieval.



Schema Version Retrieval

```
Connect to schema version 2
Perform Backward Schema Version Compatibility Algorithm
    Retrieve the structural version(Version 2) from schema version table
        Retrieve Measure sum_sale to Array
        Retrieve Measure sum_unitsales to Array
        Retrieve Measure max_sales to Array
        Retrieve Measure avg_sales to Array
        Retrieve Dimension Dtime to Array
        Retrieve Level/Attribute year to Array
        Retrieve Level/Attribute quarter to Array
        Retrieve Level/Attribute month to Array
        Retrieve Level/Attribute ddate to Array
        Retrieve Level/Attribute dtime_key to Array
        Retrieve Level/Attribute day_of_week to Array
        Retrieve Level/Attribute holiday_flag to Array
        Retrieve Dimension Product to Array
        Retrieve Level/Attribute package_size to Array
        Retrieve Level/Attribute department to Array
        Retrieve Level/Attribute category to Array
        Retrieve Level/Attribute brand to Array
    Check data management log
    Verify valid time of SV
    Rebuild fact class
    Get the specified schema version(Version 2)
    Create Tree View
    Refresh DB Seman
```

Processing completed successfully

Close

**Figure 6.5: Schema Version Retrieval Details**

According to this screen, we can check all necessary steps of schema version retrieval process starting from the current schema version to the specified schema version. This screen is very helpful for validating the logical idea of the backward schema version compatibility algorithm as well as for checking the MDB components (element of dimension and fact class) constituting the legal schema version.

Moreover, the system retains the record of schema change details in the system log. The log keeps all activities taking place with the structural schema for all versions. It also reveals the time of the sequences of the schema changes. As a consequence, the database administrator can trace evolution process taking place in any schema version. The System Log is illustrated in Figure 6.6.

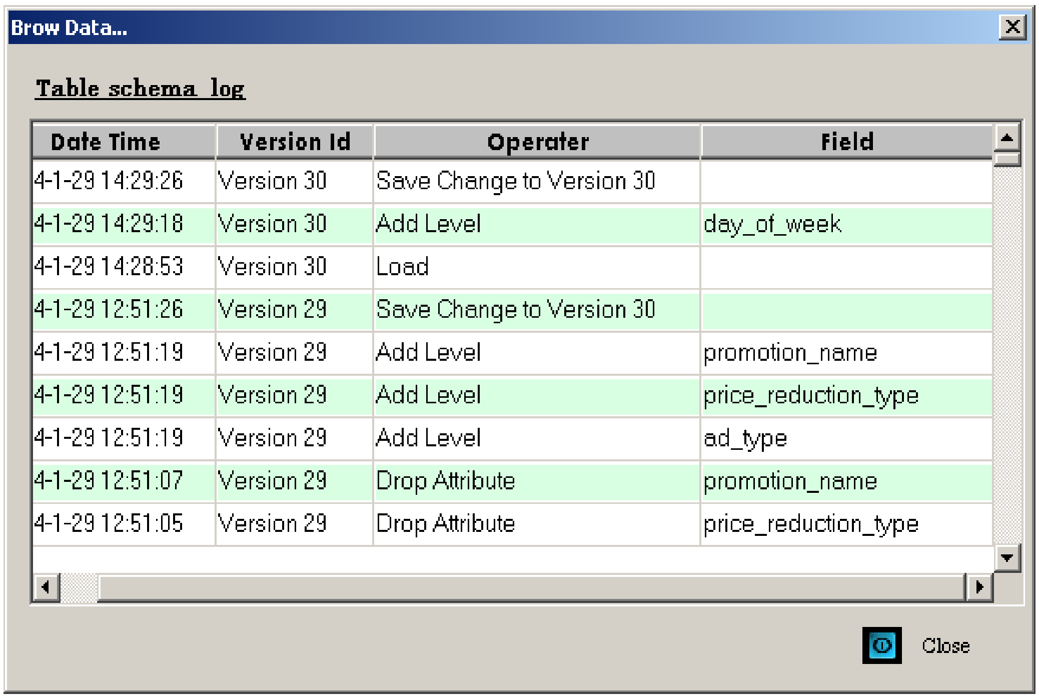


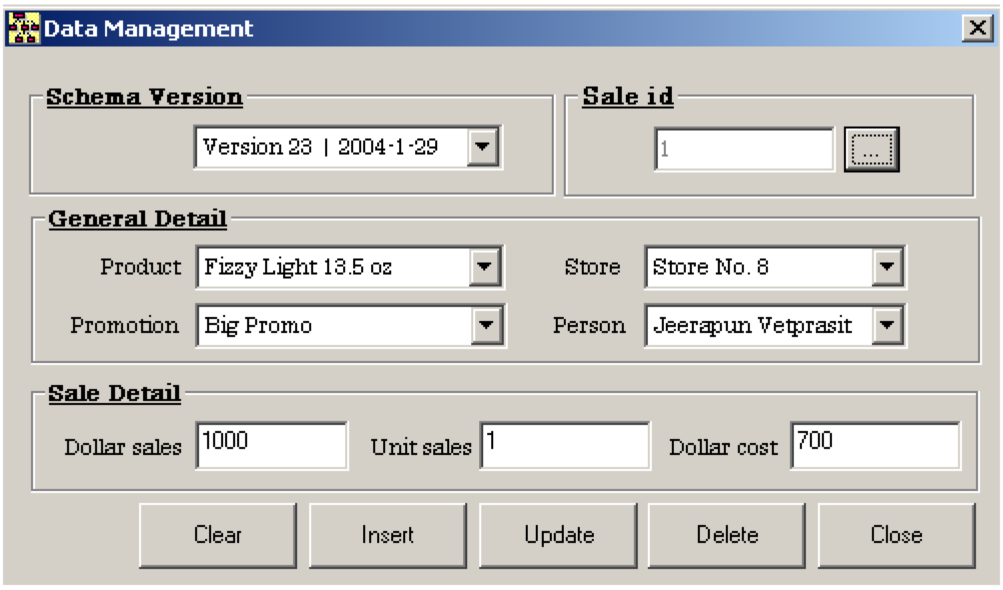
Table schema log

| Date Time | Version Id | Operater | Field |
|---|---|---|---|
| 4-1-29 14:29:26 | Version 30 | Save Change to Version 30 | |
| 4-1-29 14:29:18 | Version 30 | Add Level | day_of_week |
| 4-1-29 14:28:53 | Version 30 | Load | |
| 4-1-29 12:51:26 | Version 29 | Save Change to Version 30 | |
| 4-1-29 12:51:19 | Version 29 | Add Level | promotion_name |
| 4-1-29 12:51:19 | Version 29 | Add Level | price_reduction_type |
| 4-1-29 12:51:19 | Version 29 | Add Level | ad_type |
| 4-1-29 12:51:07 | Version 29 | Drop Attribute | promotion_name |
| 4-1-29 12:51:05 | Version 29 | Drop Attribute | price_reduction_type |

**Figure 6.6: System Log**

- **Data Management Function**

This function is developed to assist the OLAP designer/administrator to perform instance manipulation (insertion, updating and deletion) for the active schema version. Also, this feature demonstrates that the prototype serves the evolution of multidimensional instance as well. The screen design of Data Management is shown in Figure 6.7


Data Management

Schema Version: Version 23 | 2004-1-29

Sale id: 1

General Detail

Product: Fizzy Light 13.5 oz

Store: Store No. 8

Promotion: Big Promo

Person: Jeerapun Vetprasit

Sale Detail

Dollar sales: 1000

Unit sales: 1

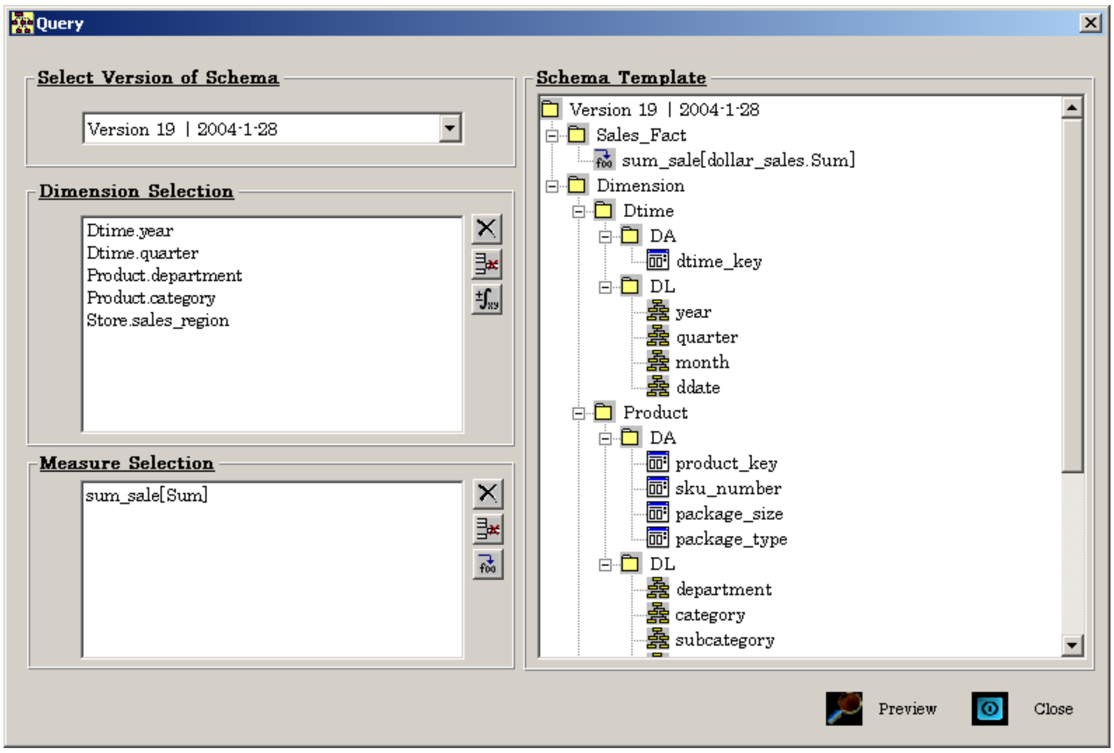
Dollar cost: 700

**Figure 6.7: Data Management Screen**

After the change has occurred, the system then generates the SQL statement that corresponds to the modification action and the affected schema version is updated accordingly. Additionally, these SQL statements are well stored and can be used as evidence in signifying the change of instance for the modified schema version.

- **Query Processing**

The multidimensional query function, which is a core engine of OLAP tool, is designed for the inquiry. It provides an easy-to-use interface. Users have to select the schema version required for an inquiry, and then the system will produce the multidimensional database schema tree with respect to the specified schema version. Apparently, users can select dimension and fact information by dragging the dimension level and measure and dropping to the reserved area of dimension and measure. In measure area, the basic query operations related to object behaviors on class “Fact” are provided. Users can change the operations and conditions via the available option menu. Finally, the query result is shown after the “Preview” button is pressed. Based on the derived cube version, users can view cube by rolling up or drilling down via the selected dimension level. Moreover, we provide the analytical spread sheet function on this screen. The result of query processing can be exported to MS.Excel for further analysis. The screens are illustrated in Figures 6.8 and 6.9.


Query

**Select Version of Schema**
Version 19 | 2004-1-28

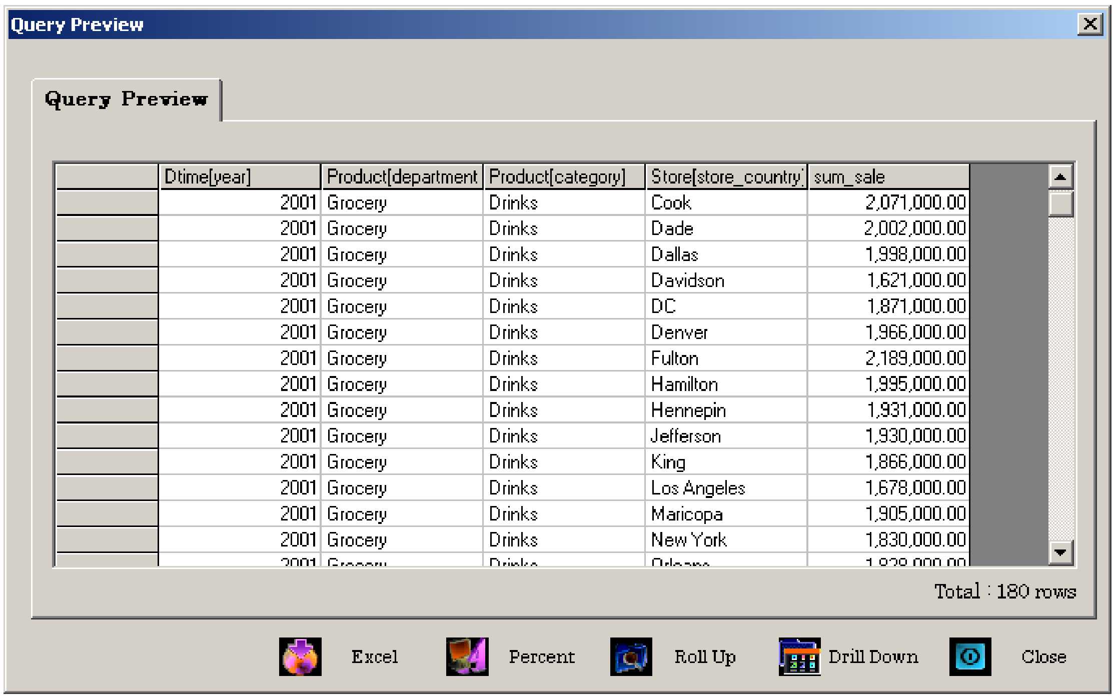
**Dimension Selection**
Dtime.year
Dtime.quarter
Product.department
Product.category
Store.sales_region

**Measure Selection**
sum_sale[Sum]

**Schema Template**
- Version 19 | 2004-1-28
  - Sales_Fact
    - sum_sale[dollar_sales.Sum]
  - Dimension
    - Dtime
      - DA
        - dtime_key
      - DL
        - year
        - quarter
        - month
        - ddate
    - Product
      - DA
        - product_key
        - sku_number
        - package_size
        - package_type
      - DL
        - department
        - category
        - subcategory

Preview Close

**Figure 6.8: Query Screen**


Query Preview

**Query Preview**

| | Dtime[year] | Product[department] | Product[category] | Store[store_country] | sum_sale |
|---|---|---|---|---|---|
| | 2001 | Grocery | Drinks | Cook | 2,071,000.00 |
| | 2001 | Grocery | Drinks | Dade | 2,002,000.00 |
| | 2001 | Grocery | Drinks | Dallas | 1,998,000.00 |
| | 2001 | Grocery | Drinks | Davidson | 1,621,000.00 |
| | 2001 | Grocery | Drinks | DC | 1,871,000.00 |
| | 2001 | Grocery | Drinks | Denver | 1,966,000.00 |
| | 2001 | Grocery | Drinks | Fulton | 2,189,000.00 |
| | 2001 | Grocery | Drinks | Hamilton | 1,995,000.00 |
| | 2001 | Grocery | Drinks | Hennepin | 1,931,000.00 |
| | 2001 | Grocery | Drinks | Jefferson | 1,930,000.00 |
| | 2001 | Grocery | Drinks | King | 1,866,000.00 |
| | 2001 | Grocery | Drinks | Los Angeles | 1,678,000.00 |
| | 2001 | Grocery | Drinks | Maricopa | 1,905,000.00 |
| | 2001 | Grocery | Drinks | New York | 1,830,000.00 |

Total : 180 rows

Excel Percent Roll Up Drill Down Close

**Figure 6.9: Query Result Preview Screen**

## 6.3 OOMDB Schema Evolution Validation

To validate all functional modules of OOMDB SEMAN, the system needs to be validated in two levels: Structural schema validation and Instance validation.

For the structural schema validation, we first establish the test case of schema evolution. In our experiment, 100 schema versions are used for the verification. Additionally, each schema version is randomly tested with the set of schema evolution operations we have defined in chapter 4. The test is thus covered by all schema evolution methods. To demonstrate the validation process of how the older schema version is changed to the newer version, the schema version transformation process screen captured from the OOMDB SEMAN prototype is presented. Obviously, we can explore the necessary steps performed by the prototype system from this screen. As a result, the details of the selective test case can be checked against the evolution tasks in the respective schema version transformation process screen. The resulting version shows the structural schema version that has been changed. By this, the result can be compared to the structural database schema presented in OOMDB SEMAN. Furthermore, we can check the result by exploring the system log archive and algorithm trace log to see whether the evolution tasks of the presented case and prototype are similar or not.

For the instance validation, the MS. Analysis Manager is used as a standard tool to validate the correctness and consistency of the functionality against the OOMDB SEMAN. In practice, we used 50 queries test scenarios that cover all cube operation types to apply to 100 schema versions.

Here, the five test cases of schema modification starting from schema versions no.1 to no. 6 are selected for a presentation here. The case presents the evolution information of the starting version and resulting version by describing their multidimensional database schema, interval time of the schema version (derived from the system log archive) and evolution operations. The example cases are shown on next pages.

## Case 1

**SV1**

DC1: Dtime
DA: Dtime_key
DL: Year, Quarter, Month, Ddate

DC2: Product
DA : Package size
DL : Department,Category,Brand

FC: Sales
FA: sum_sales, sum_unitsale

**SV2**

DC1: Dtime
DA: Dtime_key, Day_of_weeks, Holiday_flags
DL: Year,Quarter, Month, Ddate

DC2: Product
DA: Package size
DL: Department, Category, Brand

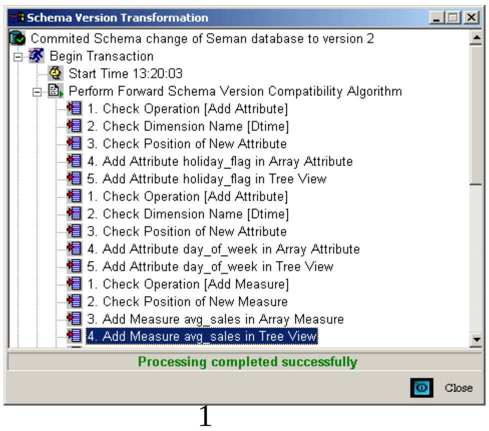
FC: Sales
FA: sum_sales, sum_unitsales, max_sales, avg_sales

$$\longrightarrow$$

[Ts, Te] = [ 1/11/03 –18:00 ,12/11/03-13:20 ]

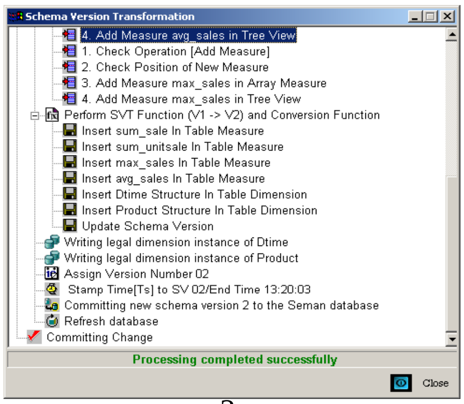
***O*$_1$**: Add_DA(Day_of_weeks.holiday_flags(Dtime)}
***O*$_2$**: Add_FA{(max_sales),(avg_sales)}

Case 1 shows the evolution details of the schema change from schema version 1 to schema version 2. The multidimensional database schema incorporating dimension classes and fact class of the starting version (version 1) and the affected version (version 2) is presented. In this case, the construction of the schema version 2 is derived from the change occurring in schema version 1. Generally, the cause of schema evolution are the evolution operations performed on the schema version 1. Here, the operations “Add_Dimension Attribute” and “Add_Fact Attribute” are the triggers of changes. According to these operations, the classification relationship is not updated since the changes affect only within their class. The process of changes is conformed to the proposed functions and algorithm in a step-by-step manner. Therefore, we present the schema transition process in terms of algorithmic way. Figure 6.10 shows the schema version transformation process (SV1→ SV2).



```
Commited Schema change of Seman database to version 2
Begin Transaction
    Start Time 13:20:03
    Perform Forward Schema Version Compatibility Algorithm
        1. Check Operation [Add Attribute]
        2. Check Dimension Name [Dtime]
        3. Check Position of New Attribute
        4. Add Attribute holiday_flag in Array Attribute
        5. Add Attribute holiday_flag in Tree View
        1. Check Operation [Add Attribute]
        2. Check Dimension Name [Dtime]
        3. Check Position of New Attribute
        4. Add Attribute day_of_week in Array Attribute
        5. Add Attribute day_of_week in Tree View
        1. Check Operation [Add Measure]
        2. Check Position of New Measure
        3. Add Measure avg_sales in Array Measure
        4. Add Measure avg_sales in Tree View
Processing completed successfully
```

1



```
        4. Add Measure avg_sales in Tree View
        1. Check Operation [Add Measure]
        2. Check Position of New Measure
        3. Add Measure max_sales in Array Measure
        4. Add Measure max_sales in Tree View
    Perform SVT Function (V1 -> V2) and Conversion Function
        Insert sum_sale In Table Measure
        Insert sum_unitsale In Table Measure
        Insert max_sales In Table Measure
        Insert avg_sales In Table Measure
        Insert Dtime Structure In Table Dimension
        Insert Product Structure In Table Dimension
        Update Schema Version
    Writing legal dimension instance of Dtime
    Writing legal dimension instance of Product
    Assign Version Number 02
     Stamp Time[Ts] to SV 02/End Time 13:20:03
    Committing new schema version 2 to the Seman database
    Refresh database
Committing Change
Processing completed successfully
```

2

**Figure 6.10: Schema Version Transformation Process (SV1→SV2)**

## Case 2

**SV2**

DC1: Dtime
DA: Dtime_key, Day_of_weeks, Holiday_flags
DL: Year,Quarter, Month, Ddate

DC2: Product
DA: Package size
DL: Department, Category, Brand

FC: Sales
FA: sum_sales, sum_unitsales, max_sales, avg_sales

**SV3**

DC1: Dtime
DA: Dtime_key, Day_of_weeks, Holiday_flags
DL: Year,Quarter, Month, Ddate

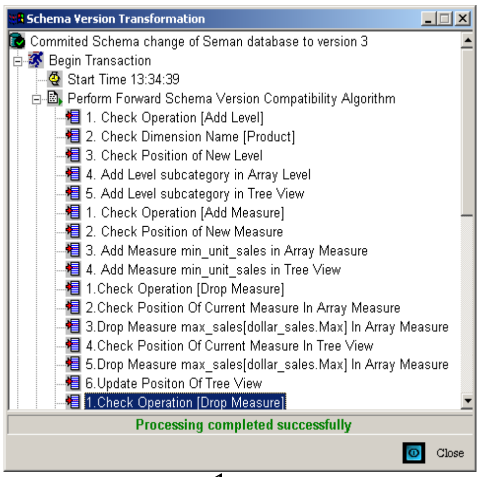
DC2: Product
DA: Package size
DL: Department, Category, Subcategory, Brand

FC: Sales
FA: sum_sales, min_unit_sales

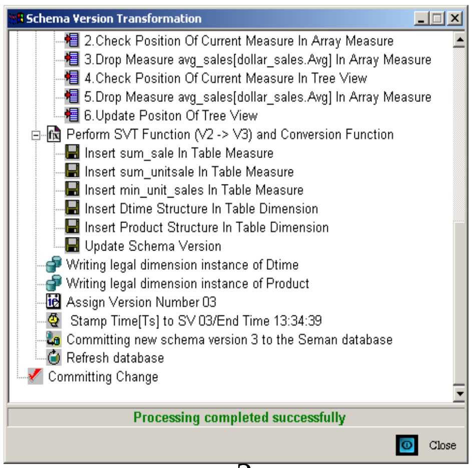
[Ts, Te] = [12/11/03-13:20 ,15/11/03-13:34 ]

***O***$_1$: Add_level(Subcategory(Product){Category,Brand})
***O***$_2$: Add_FA{(min_unit_sales)}
***O***$_3$: Del_FA{(max_sales),(avg_sales)}

Case 2 shows the evolution details of the schema change from schema version 2 to schema version 3. The evolution operations "Add DL", "Add FA", and "Del FA" are applied to schema version 2. This yields changes of structural schema and instances of DC and FC. With the "Add DL" operation, the classification relationship of the affected class is also updated. Figure 6.11 presents the schema version evolution process with respect to the evolution operations depicted in the case.


Schema Version Transformation
Commited Schema change of Seman database to version 3
Begin Transaction
Start Time 13:34:39
Perform Forward Schema Version Compatibility Algorithm
1. Check Operation [Add Level]
2. Check Dimension Name [Product]
3. Check Position of New Level
4. Add Level subcategory in Array Level
5. Add Level subcategory in Tree View
1. Check Operation [Add Measure]
2. Check Position of New Measure
3. Add Measure min_unit_sales in Array Measure
4. Add Measure min_unit_sales in Tree View
1.Check Operation [Drop Measure]
2.Check Position Of Current Measure In Array Measure
3.Drop Measure max_sales[dollar_sales.Max] In Array Measure
4.Check Position Of Current Measure In Tree View
5.Drop Measure max_sales[dollar_sales.Max] In Array Measure
6.Update Positon Of Tree View
1.Check Operation [Drop Measure]
Processing completed successfully

1


Schema Version Transformation
2.Check Position Of Current Measure In Array Measure
3.Drop Measure avg_sales[dollar_sales.Avg] In Array Measure
4.Check Position Of Current Measure In Tree View
5.Drop Measure avg_sales[dollar_sales.Avg] In Array Measure
6.Update Positon Of Tree View
Perform SVT Function (V2 -> V3) and Conversion Function
Insert sum_sale In Table Measure
Insert sum_unitsale In Table Measure
Insert min_unit_sales In Table Measure
Insert Dtime Structure In Table Dimension
Insert Product Structure In Table Dimension
Update Schema Version
Writing legal dimension instance of Dtime
Writing legal dimension instance of Product
Assign Version Number 03
Stamp Time[Ts] to SV 03/End Time 13:34:39
Committing new schema version 3 to the Seman database
Refresh database
Committing Change
Processing completed successfully

2

**Figure 6.11: Schema Version Transformation Process (SV2→SV3)**

## Case 3

**SV3**

DC1: Dtime
DA: Dtime_key, Day_of_weeks, Holiday_flags
DL: Year,Quarter, Month, Ddate

DC2: Product
DA: Package size
DL: Department, Category, Subcategory, Brand

FC: Sales
FA: sum_sales, min_unit_sales

[Ts, Te] = [15/11/03-13:34 ,24/11/03-14:54 ]

**SV4**

DC1: Dtime
DA: Dtime_key, Day_of_weeks, Holiday_flags
DL: Year,Quarter, Month, Ddate

DC2: Product
DA: Package size
DL: Department, Category, Subcategory, Brand

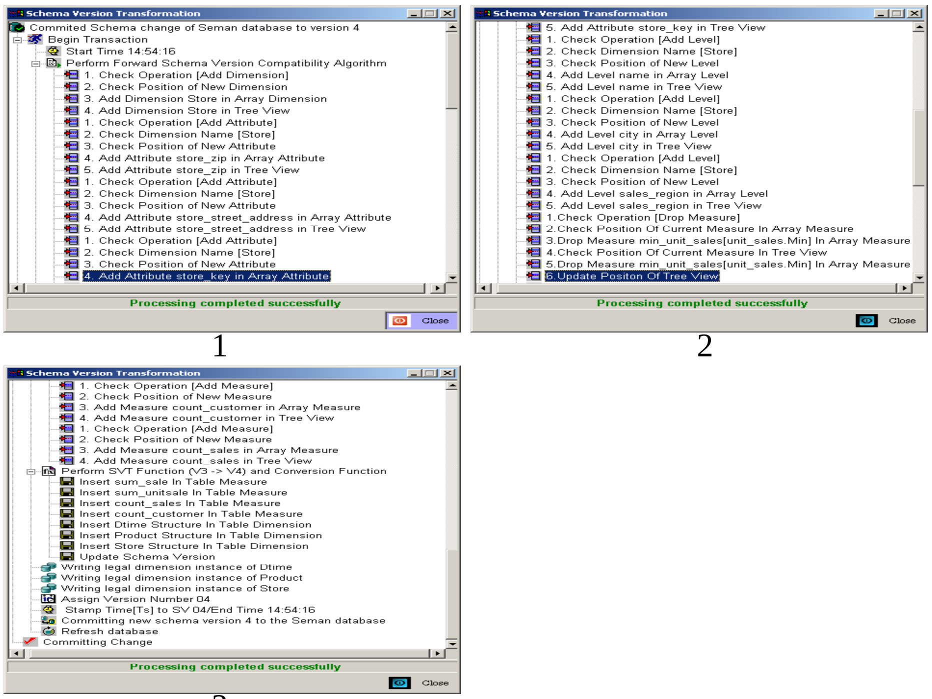
DC3: Store
DA: Store_key, Store_street_address, Store_zip
DL: Sales_region, City, Name

FC: Sales
FA: sum_sales, count_sales, count_customer, avg_sales

$$\xrightarrow{\hspace{6cm}}$$

***$O_1$***: Add_DC (Store)
***$O_2$***: Del_FA{(min_unit_sales)}
***$O_3$***: Add_FA{(count_sales), (count_customer),(avg_sales)}

Case 3 shows the evolution details of the schema change from schema version 3 to schema version 4. Here, the operations "Add DC", "Del FA" and "Add FA" are applied to the schema version 3. With the "Add DC" operation, the set of DC in the resulting version: SV4, is updated and the granularity of FC is also affected. Figure 6.12 presents the schema version evolution process with respect to the evolution operations depicted in the case.



```
Schema Version Transformation
Commited Schema change of Seman database to version 4
  Begin Transaction
    Start Time 14:54:16
    Perform Forward Schema Version Compatibility Algorithm
      1. Check Operation [Add Dimension]
      2. Check Position of New Dimension
      3. Add Dimension Store in Array Dimension
      4. Add Dimension Store in Tree View
      1. Check Operation [Add Attribute]
      2. Check Dimension Name [Store]
      3. Check Position of New Attribute
      4. Add Attribute store_zip in Array Attribute
      5. Add Attribute store_zip in Tree View
      1. Check Operation [Add Attribute]
      2. Check Dimension Name [Store]
      3. Check Position of New Attribute
      4. Add Attribute store_street_address in Array Attribute
      5. Add Attribute store_street_address in Tree View
      1. Check Operation [Add Attribute]
      2. Check Dimension Name [Store]
      3. Check Position of New Attribute
      4. Add Attribute store_key in Array Attribute
Processing completed successfully
Close
```

1

```
Schema Version Transformation
      5. Add Attribute store_key in Tree View
      1. Check Operation [Add Level]
      2. Check Dimension Name [Store]
      3. Check Position of New Level
      4. Add Level name in Array Level
      5. Add Level name in Tree View
      1. Check Operation [Add Level]
      2. Check Dimension Name [Store]
      3. Check Position of New Level
      4. Add Level city in Array Level
      5. Add Level city in Tree View
      1. Check Operation [Add Level]
      2. Check Dimension Name [Store]
      3. Check Position of New Level
      4. Add Level sales_region in Array Level
      5. Add Level sales_region in Tree View
      1.Check Operation [Drop Measure]
      2.Check Position Of Current Measure In Array Measure
      3.Drop Measure min_unit_sales[unit_sales.Min] In Array Measure
      4.Check Position Of Current Measure In Tree View
      5.Drop Measure min_unit_sales[unit_sales.Min] In Array Measure
      6.Update Positon Of Tree View
Processing completed successfully
Close
```

2

```
Schema Version Transformation
      1. Check Operation [Add Measure]
      2. Check Position of New Measure
      3. Add Measure count_customer in Array Measure
      4. Add Measure count_customer in Tree View
      1. Check Operation [Add Measure]
      2. Check Position of New Measure
      3. Add Measure count_sales in Array Measure
      4. Add Measure count_sales in Tree View
    Perform SVT Function (V3 -> V4) and Conversion Function
      Insert sum_sale In Table Measure
      Insert sum_unitsale In Table Measure
      Insert count_sales In Table Measure
      Insert count_customer In Table Measure
      Insert Dtime Structure In Table Dimension
      Insert Product Structure In Table Dimension
      Insert Store Structure In Table Dimension
      Update Schema Version
    Writing legal dimension instance of Dtime
    Writing legal dimension instance of Product
    Writing legal dimension instance of Store
    Assign Version Number 04
     Stamp Time[Ts] to SV 04/End Time 14:54:16
    Committing new schema version 4 to the Seman database
    Refresh database
  Committing Change
Processing completed successfully
Close
```

3

**Figure 6.12: Schema Version Transformation Process (SV3→SV4)**

## Case 4

**SV4**

DC1: Dtime
DA: Dtime_key, Day_of_weeks, Holiday_flags
DL: Year,Quarter, Month, Ddate

DC2: Product
DA: Package size
DL: Department, Category, Subcategory, Brand

DC3: Store
DA: Store_key, Store_street_address, Store_zip
DL: Sales_region, City, Name

FC: Sales
FA: sum_sales, count_sale, count_customer, avg_sales

**SV5**

DC1: Dtime
DA: Dtime_key, Day_of_weeks, Holiday_flags
DL: Year, Month, Ddate

DC2: Product
DA: Package size
DL: Department, Category, Subcategory, Brand

DC3: Store
DA: Store_key, Store_street_address
DL: Store_country, Sales_region, City, Name

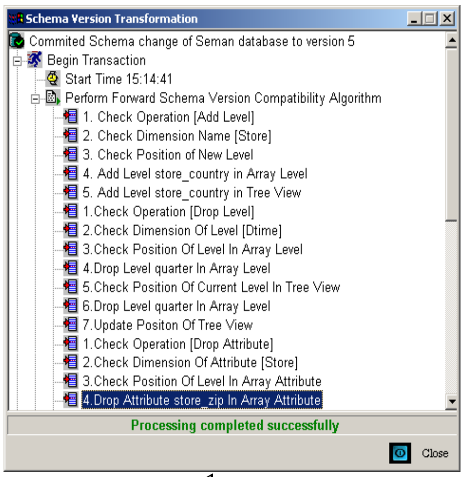
FC: Sales
FA: sum_sales, count_sale, count_customer, avg_sales

[Ts, Te] = [24/11/03-14:54 ,24/11/03-15:14 ]

$\longrightarrow$

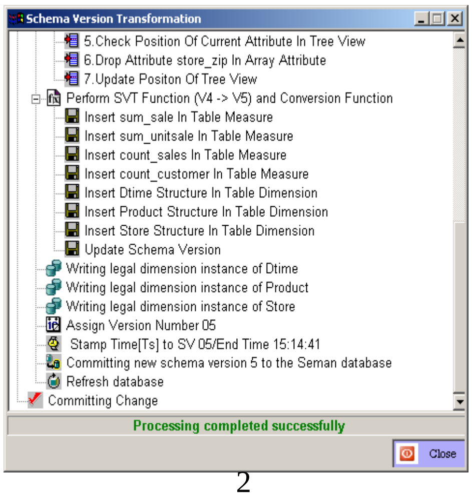
***O*₁**: Add_level(Store_country(Store){All, Sales_region})
***O*₂**: Del_level(Quarter(Dtime))
***O*₂**: Del_DA(Store_zip(Store))

Case 4 shows the evolution details of the schema change from schema version 4 to schema version 5. In this case, the operations "Add DL", "Del DL" , and "Del DA" are the triggers of changes. These operations are applied to 2 DCs: Store and Dtime. The change not only affects their intra-class, but also the associative classification relationship. Figure 6.13 presents the schema version evolution process with respect to the evolution operations depicted in the case.


Schema Version Transformation

```
Commited Schema change of Seman database to version 5
Begin Transaction
    Start Time 15:14:41
    Perform Forward Schema Version Compatibility Algorithm
        1. Check Operation [Add Level]
        2. Check Dimension Name [Store]
        3. Check Position of New Level
        4. Add Level store_country in Array Level
        5. Add Level store_country in Tree View
        1.Check Operation [Drop Level]
        2.Check Dimension Of Level [Dtime]
        3.Check Position Of Level In Array Level
        4.Drop Level quarter In Array Level
        5.Check Position Of Current Level In Tree View
        6.Drop Level quarter In Array Level
        7.Update Positon Of Tree View
        1.Check Operation [Drop Attribute]
        2.Check Dimension Of Attribute [Store]
        3.Check Position Of Level In Array Attribute
        4.Drop Attribute store_zip In Array Attribute
Processing completed successfully
```

1


Schema Version Transformation

```
        5.Check Position Of Current Attribute In Tree View
        6.Drop Attribute store_zip In Array Attribute
        7.Update Positon Of Tree View
    Perform SVT Function (V4 -> V5) and Conversion Function
        Insert sum_sale In Table Measure
        Insert sum_unitsale In Table Measure
        Insert count_sales In Table Measure
        Insert count_customer In Table Measure
        Insert Dtime Structure In Table Dimension
        Insert Product Structure In Table Dimension
        Insert Store Structure In Table Dimension
        Update Schema Version
    Writing legal dimension instance of Dtime
    Writing legal dimension instance of Product
    Writing legal dimension instance of Store
    Assign Version Number 05
    Stamp Time[Ts] to SV 05/End Time 15:14:41
    Committing new schema version 5 to the Seman database
    Refresh database
Committing Change
Processing completed successfully
```

2

**Figure 6.13: Schema Version Transformation Process (SV4→SV5)**

## Case 5

**SV5**

DC1: Dtime
DA: Dtime_key, Day_of_weeks, Holiday_flags
DL: Year, Month, Ddate

DC2: Product
DA: Package size
DL: Department, Category, Subcategory, Brand

DC3: Store
DA: Store_key, Store_street_address
DL: Store_country, Sales_region, City, Name

FC: Sales
FA: sum_sales, count_sale, count_customer, avg_sales

**SV6**

DC1: Dtime
DA: Dtime_key, Day_of_weeks, Holiday_flags
DL: Year, Month, Ddate

DC2: Product
DA: Package size
DL: Department, Category, Subcategory, Brand

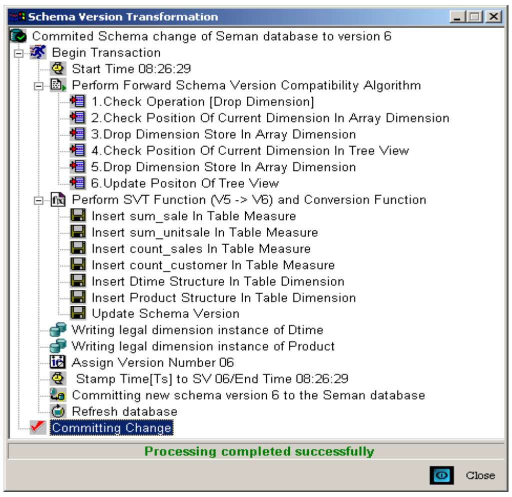
FC: Sales
FA: sum_sales, count_sale, count_customer, avg_sales

[Ts, Te] = [24/11/03-15:14 ,30/11/03-08:26 ]

$\longrightarrow$

$O_1$: Delete DC(Store)

Case 5 shows the evolution details of the schema change from schema version 5 to schema version 6. In this case, the "Delete DC" operation is applied for changes. The changes affect the deletion of dimension class "Store", thus the schema and its instance do not appear in next versions. By this effect, the loss of instance is versioned and maintained. Also, the fact schema needs to be restructured. Figure 6.14 presents the schema version evolution process with respect to the evolution operations depicted in the case.


Schema Version Transformation

```
Commited Schema change of Seman database to version 6
  Begin Transaction
    Start Time 08:26:29
    Perform Forward Schema Version Compatibility Algorithm
      1.Check Operation [Drop Dimension]
      2.Check Position Of Current Dimension In Array Dimension
      3.Drop Dimension Store In Array Dimension
      4.Check Position Of Current Dimension In Tree View
      5.Drop Dimension Store In Array Dimension
      6.Update Positon Of Tree View
    Perform SVT Function (V5 -> V6) and Conversion Function
      Insert sum_sale In Table Measure
      Insert sum_unitsale In Table Measure
      Insert count_sales In Table Measure
      Insert count_customer In Table Measure
      Insert Dtime Structure In Table Dimension
      Insert Product Structure In Table Dimension
      Update Schema Version
    Writing legal dimension instance of Dtime
    Writing legal dimension instance of Product
    Assign Version Number 06
     Stamp Time[Ts] to SV 06/End Time 08:26:29
    Committing new schema version 6 to the Seman database
    Refresh database
  Committing Change
Processing completed successfully
```

**Figure 6.14: Schema Version Transformation Process (SV5→SV6)**

In summary, we can assure that the modification of schema version is monitored and controlled by our proposed functions and algorithm since the process reveals the change impact of the affected object or class elements of the particular schema version correctly. Generally speaking, the result explicitly depicts the correctness of the transformation of all structural schema versions. Actually, the evolution process totally supports the consistency and correctness in both the schema and instance level. However, the validation of the above test case and the schema version transformation process generated by the system does not still apparently show the support of instance propagation control. To substantiate the MD instance validation, we use the query test to substantiate the verification.

## 6.4 Query Testing

To verify the correctness of OOMDB SEMAN functionality in the instance level, 50 query scenarios are set up to test with 100 schema versions. To define the set of test queries, we adopt some query statements from [19]. The query scenarios are grouped into three categories based on multidimensional operations: data slicing, data dicing and hierarchical operations. The result of the 50 identical queries is applied to both our developed tool and the commercial OLAP tool: Microsoft SQL Server. Then we compare the result from these two systems.

Below are the query scenarios for the experiment.

Data Slicing

1. Find sum of dollar sales by product and year
2. Find average dollar_sales by product and date
3. Find maximum number of dollar_sales by product category
4. Find minimum number of dollar_sales by product, promotion type and store
5. Count number of sales transactions by date, product and store
6. Find average of dollar sales by year 2001 and 2002
7. Find sum of dollar sales by salesperson, store and product
8. Find minimum unit sales and maximum dollar cost by product in subcategory and brand

9. Find the difference between sum of dollar sales and dollar cost by store, product and date
10. Find the difference in number of customers in October 2001 and December 2002 by store and product
11. Find sum of dollar sales by product, promotion type and store in 2000 and 2003
12. Find percentage of dollar sales to dollar cost by store, product and promotion type
13. Find the percentage of customers between November 2002 and December 2002 by store and product
14. Find the average of unit sales by product, promotion type, store and date
15. Find sum of unit sales by promotion type, store, product and date

Data Dicing

16. Find sum of dollar sales for product of all categories which were sold in New York
17. Find average number of customers who bought products “Athletic Drink”, “Clear Refresher”, “Fizzy Classic”, “fizzy Light” or “Strong Cola” during Oct 1- 7, 2000
18. Count number of customers influenced by promotion type “Big Promo” or “Blue Ribbon Discounts”
19. Find minimum unit sales for product “Athletic Drink” or “Strong Cola”, using the promotion type “Two for One” in stores No.1 to 10
20. Count number of sales transactions by date, product and store for product category of Drink
21. Rank the unit sales by store and product for products “Lasagna”, “Beef Stew”, “Turkey Dinner” and “Chicken Dinner” which were sold in stores No. 1 to 5
22. Find sum of dollar sales and unit sales for products “Athletic Drink”, “Clear Refresher”, “Dry Tissues”, “Fizzy Classic”, “Fizzy Light”, “Paper Towels” or “Strong Cola” and “Wet Wipes”

23. Count customer numbers who bought products “Lasagna” or “Beef Stew” from either stores No. 1 to 9 during Nov 10- 20, 2001
24. Find the difference between sum of dollar sales and dollar cost by store, product and date for stores No. 2, 11, 12, 13, 16 or 20, and products “Buffalo Jerky”, “Dried Grits”, Onion Slices”, “Power chips” or “Salty Corn”
25. Find the difference in number of customers in October 2000 and in December 2000 by store and product for products “Lasagna”, “Beef Stew”, “Extra Nougat”, “Lots of Nuts” or “Sweet Tooth”, which were sold in stores No. 5 to 9
26. Find the percentage of each unit sales to total unit sales by product and promotion type for products “Buffalo Jerky”, “Dried Grits”, “Onion Slices”, “Power Chips” or “Salty Corn” and promotion types “POS Grabbers”, “Redemption” or “Big Promo”
27. Find the percentage of dollar sales to dollar cost by store, product and promotion type for products “Fizzy Light”, “Fizzy Classic” or “Athletic Drink”, which were sold in stores No. 2, 11, 12, 13, 16 or 20, using promotion types “Two for One” or “Shelf Talkers”
28. Find the percentage of customers in October 2001 to in December 2001 by store and product for stores No. 1, 2, 3, 7 or 15 and product “Onion Slices” or buffalo Jerky”
29. Find the average number of unit sales by product, promotion type and date for products “Buffalo Jerky”, “Dried Grits”, “Onion Slices”, “Power Chips” or “Salty Corn” sold in December 2000 and using promotion type “Redemption”
30. Find the unit sales by promotion type, store, product and date for the promotion type “No promotion” of products “Athletic Drink”, “Clear Refresher” and “Strong Cola”

Hierarchical operation

31. Find sum of dollar sales by category and sales city is Cook
32. Find average number of customers by brand and month
33. Find maximum number of customers by store state and advertising type
34. Find maximum number of customers by store country and advertising type

35. Find minimum unit sales by month, category, subcategory and floor plan type
36. Count number of sales transactions by day of week, sales region and subcategory
37. Rank the unit sales by store state
38. Find sum of dollars sales by category, display type and city
39. Find sum of dollar sales by sales person, customer, and store for the years 2001, 2002, and 2003
40. Find sum of unit sales and dollar sales by brand, city in Quarter 4, 2001 and 2002
41. Find the difference between dollar sales and dollar cost by floor plan type, subcategory and day of week
42. Find the difference in number of customers from October 1999 to December 1999 by brand and advertising type
43. Find the percentage of dollar sales to dollar cost by store state, department and day of week
44. Find sum of dollar sales for category “Drinks” or “Supplies”, which were sold in sales region “Eastern”
45. Find average number of customers who bought product with brand “Frozen Bird” or “American Corn” in Oct’01 and Oct’02
46. Find maximum number of customers in state “PA” or “CA”, who were influenced by advertising type “Daily Paper”
47. Find minimum unit sales in Nov’02 for product in category “Drinks” or “Food”, which were sold in store with floor plan type “Original”
48. Rank the unit sales for state “LA”, “TX” and “PA”
49. Find the percentage of dollar sales to dollar cost of store in state “CA” or “PA”, which sold the product in department “Grocery” on Wednesday
50. Find the average number of unit sales for product in category “Drinks” or “Supplies”, sold in sales region “Mid West” in Oct’2003

## 6.5 Result of the Experiment

This section shows the experimental result that the overall function of OOMDB SEMAN is correct. By the concept, we use a set of queries defined in section 6.4 to perform the test against OOMDB SEMAN and Analysis Services of MS. SQL Server 2000. The test is performed on the various available schema versions by the different queries. In practice, the verification is performed version by version. The multidimensional database schema that corresponds to the database schema in OOMDB SEMAN is manually created for the Analysis Services. Therefore, the same query is verified to the same structural database schema of the OOMDB SEMAN and Analysis Services. In addition, the query result derived from the system prototype can be exported to the MS.Excel for further analysis. Although some operations may not be provided by the prototype of MS. Analysis Manager, the MS. Excel can provide a solution. We also check the consistency of the query result in forward and backward manner. The results of the verification of both systems reveal that the proposed OOMDB SEMAN functionalities are correct and consistent in all versions. The evolution method is simply illustrated in Figure 6.15

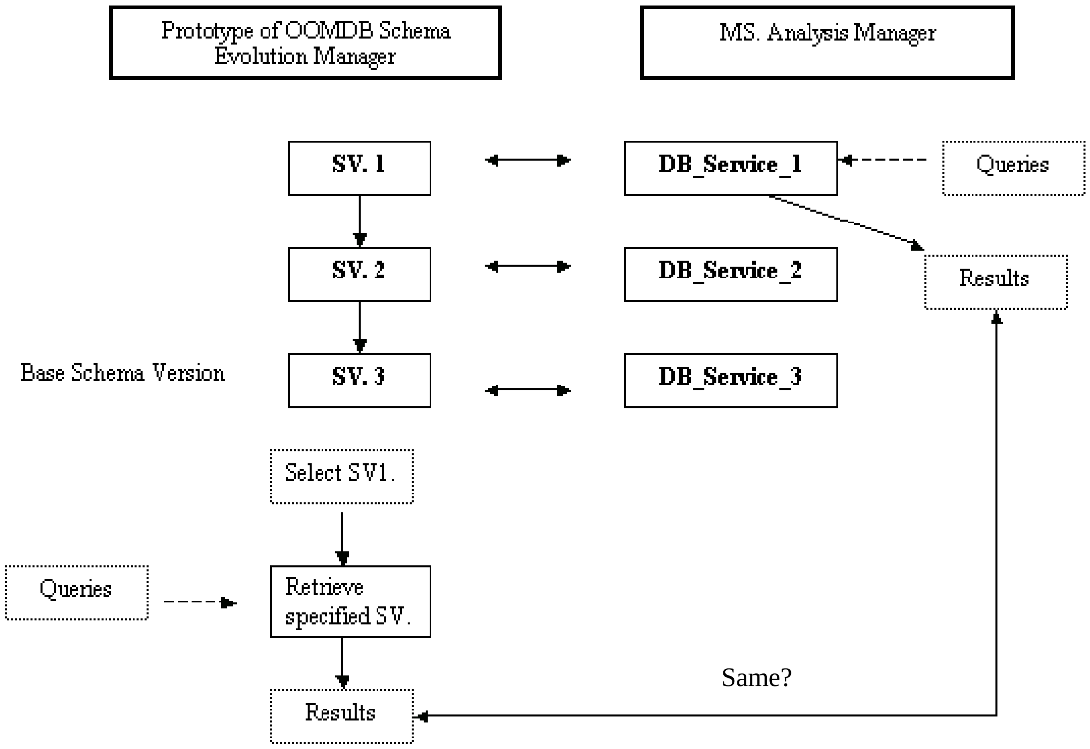


**Figure 6.15: Validation Method**

Some examples of the result are presented as follows:

**Schema Version No. 1**

**Sum of dollar sales by product and year**

**OOMDB SEMAN**

| Year | Product_ Department | Sum_of_ dollar sales |
|---|---|---|
| 2001 | Grocery | 128,145,000.00 |
| 2001 | Household | 22,152,000.00 |
| 2002 | Grocery | 129,877,000.00 |
| 2002 | Household | 22,758,000.00 |
| 2003 | Grocery | 125,398,000.00 |
| 2003 | Household | 22,118,000.00 |

**MS. Analysis Manager**

| Year | Product_ Department | Sum_of_ dollar sales |
|---|---|---|
| 2001 | Grocery | 128,145,000.00 |
| 2001 | Household | 22,152,000.00 |
| 2002 | Grocery | 129,877,000.00 |
| 2002 | Household | 22,758,000.00 |
| 2003 | Grocery | 125,398,000.00 |
| 2003 | Household | 22,118,000.00 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

**Sum of unit sales by product category is “Drink” in 4th Quarter for 2001 to 2003**

**OOMDB SEMAN**

| Year | Quarter | Product_ Category | Sum_of_ Unitsales |
|---|---|---|---|
| 2001 | 4 | Drinks | 10,522.00 |
| 2002 | 4 | Drinks | 9,763.00 |
| 2003 | 4 | Drinks | 9,519.00 |

**MS. Analysis Manager**

| Year | Quarter | Product_ Category | Sum_of_ Unitsales |
|---|---|---|---|
| 2001 | 4 | Drinks | 10,522.00 |
| 2002 | 4 | Drinks | 9,763.00 |
| 2003 | 4 | Drinks | 9,519.00 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |

**Schema Version No.2**

**Average of dollar_sales by product and quarter**

**OOMDB SEMAN**

| Year | Quarter | Product_ Department | Average_of_ dollar sales |
|---|---|---|---|
| 2001 | 1 | Grocery | 5,509.54 |
| 2001 | 1 | Household | 5,359.36 |
| 2001 | 2 | Grocery | 5,516.27 |
| 2001 | 2 | Household | 5,321.15 |
| 2001 | 3 | Grocery | 5,427.35 |
| 2001 | 3 | Household | 5,586.56 |
| 2001 | 1 | Grocery | 5,509.54 |
| 2001 | 4 | Grocery | 5,468.50 |
| 2001 | 4 | Household | 5,521.46 |
| 2002 | 1 | Grocery | 5,483.32 |
| 2002 | 1 | Household | 5,479.13 |
| 2002 | 2 | Grocery | 5,471.70 |
| 2002 | 2 | Household | 5,611.59 |
| 2002 | 3 | Grocery | 5,495.98 |
| 2002 | 3 | Household | 5,579.10 |
| 2002 | 4 | Grocery | 5,445.85 |
| 2002 | 4 | Household | 5,518.39 |
| 2003 | 1 | Grocery | 5,524.40 |
| 2003 | 1 | Household | 5,467.04 |
| 2003 | 2 | Grocery | 5,466.38 |
| 2003 | 2 | Household | 5,541.21 |
| 2003 | 3 | Grocery | 5,519.83 |
| 2003 | 3 | Household | 5,471.97 |
| 2003 | 4 | Grocery | 5,523.00 |
| 2003 | 4 | Household | 5,514.95 |

**MS. Analysis Manager**

| Year | Quarter | Product_ Department | Average_of_ dollar sales |
|---|---|---|---|
| 2001 | 1 | Grocery | 5,509.54 |
| 2001 | 1 | Household | 5,359.36 |
| 2001 | 2 | Grocery | 5,516.27 |
| 2001 | 2 | Household | 5,321.15 |
| 2001 | 3 | Grocery | 5,427.35 |
| 2001 | 3 | Household | 5,586.56 |
| 2001 | 1 | Grocery | 5,509.54 |
| 2001 | 4 | Grocery | 5,468.50 |
| 2001 | 4 | Household | 5,521.46 |
| 2002 | 1 | Grocery | 5,483.32 |
| 2002 | 1 | Household | 5,479.13 |
| 2002 | 2 | Grocery | 5,471.70 |
| 2002 | 2 | Household | 5,611.59 |
| 2002 | 3 | Grocery | 5,495.98 |
| 2002 | 3 | Household | 5,579.10 |
| 2002 | 4 | Grocery | 5,445.85 |
| 2002 | 4 | Household | 5,518.39 |
| 2003 | 1 | Grocery | 5,524.40 |
| 2003 | 1 | Household | 5,467.04 |
| 2003 | 2 | Grocery | 5,466.38 |
| 2003 | 2 | Household | 5,541.21 |
| 2003 | 3 | Grocery | 5,519.83 |
| 2003 | 3 | Household | 5,471.97 |
| 2003 | 4 | Grocery | 5,523.00 |
| 2003 | 4 | Household | 5,514.95 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

**Schema Version No.3**

**Minimum unit sales and maximum dollar cost by product subcategory and brand**

**OOMDB SEMAN**

| Product_ Category | Product_ Brand | Minimum_ of_unit sales |
|---|---|---|
| Drinks | Big Can | 1 |
| Drinks | National Bottle | 1 |
| Food | American Corn | 1 |
| Food | Chewy Industries | 1 |
| Food | Cold Gourmet | 1 |
| Food | Frozen Bird | 1 |
| Food | Western Vegetable | 1 |

**MS. Analysis Manager**

| Product_ Category | Product_ Brand | Minimum_ of_unit sales |
|---|---|---|
| Drinks | Big Can | 1 |
| Drinks | National Bottle | 1 |
| Food | American Corn | 1 |
| Food | Chewy Industries | 1 |
| Food | Cold Gourmet | 1 |
| Food | Frozen Bird | 1 |
| Food | Western Vegetable | 1 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

**Sum of dollar sales for category “Drinks” or “Supplies”, which were sold in 2003**

**OOMDB SEMAN**

| Year | Product_Category | Sum_of_ dollar sales |
|---|---|---|
| 2003 | Drinks | 36,765,000.00 |
| 2003 | Supplies | 22,118,000.00 |

**MS. Analysis Manager**

| Year | Product_Category | Sum_of_ dollar sales |
|---|---|---|
| 2003 | Drinks | 36,765,000.00 |
| 2003 | Supplies | 22,118,000.00 |

| Difference |
|---|
| 0 |
| 0 |

**Schema Version No.4**

**Count number of sales transactions by store name and product department**

**OOMDB SEMAN**

| Store Name | Product_ Department | Count_No._of_ Sales transaction |
|---|---|---|
| Store No. 1 | Grocery | 3,513.00 |
| Store No. 1 | Household | 614 |
| Store No. 10 | Grocery | 3,471.00 |
| Store No. 10 | Household | 586 |
| Store No. 11 | Grocery | 3,553.00 |
| Store No. 11 | Household | 618 |
| Store No. 12 | Grocery | 3,515.00 |
| Store No. 12 | Household | 612 |
| Store No. 13 | Grocery | 3,489.00 |
| Store No. 13 | Household | 629 |
| Store No. 14 | Grocery | 3,490.00 |
| Store No. 14 | Household | 597 |
| Store No. 15 | Grocery | 3,531.00 |
| Store No. 15 | Household | 604 |
| Store No. 16 | Grocery | 3,451.00 |
| Store No. 16 | Household | 602 |
| Store No. 17 | Grocery | 3,430.00 |
| Store No. 17 | Household | 547 |
| Store No. 18 | Grocery | 3,534.00 |
| Store No. 18 | Household | 636 |

**MS. Analysis Manager**

| Store Name | Product_ Department | Count_No._of_ Sales transaction |
|---|---|---|
| Store No. 1 | Grocery | 3,513.00 |
| Store No. 1 | Household | 614 |
| Store No. 10 | Grocery | 3,471.00 |
| Store No. 10 | Household | 586 |
| Store No. 11 | Grocery | 3,553.00 |
| Store No. 11 | Household | 618 |
| Store No. 12 | Grocery | 3,515.00 |
| Store No. 12 | Household | 612 |
| Store No. 13 | Grocery | 3,489.00 |
| Store No. 13 | Household | 629 |
| Store No. 14 | Grocery | 3,490.00 |
| Store No. 14 | Household | 597 |
| Store No. 15 | Grocery | 3,531.00 |
| Store No. 15 | Household | 604 |
| Store No. 16 | Grocery | 3,451.00 |
| Store No. 16 | Household | 602 |
| Store No. 17 | Grocery | 3,430.00 |
| Store No. 17 | Household | 547 |
| Store No. 18 | Grocery | 3,534.00 |
| Store No. 18 | Household | 636 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

**OOMDB SEMAN**

| Store Name | Product_ Department | Count_No._of_ Sales transaction |
|---|---|---|
| Store No. 19 | Grocery | 3,595.00 |
| Store No. 19 | Household | 596 |
| Store No. 2 | Grocery | 3,441.00 |
| Store No. 2 | Household | 599 |
| Store No. 20 | Grocery | 3,518.00 |
| Store No. 20 | Household | 632 |
| Store No. 3 | Grocery | 3,508.00 |
| Store No. 3 | Household | 562 |
| Store No. 4 | Grocery | 3,394.00 |
| Store No. 4 | Household | 628 |
| Store No. 5 | Grocery | 3,416.00 |
| Store No. 5 | Household | 641 |
| Store No. 6 | Grocery | 3,604.00 |
| Store No. 6 | Household | 650 |
| Store No. 7 | Grocery | 3,524.00 |
| Store No. 7 | Household | 632 |
| Store No. 8 | Grocery | 3,436.00 |
| Store No. 8 | Household | 595 |
| Store No. 9 | Grocery | 3,466.00 |
| Store No. 9 | Household | 608 |

**MS. Analysis Manager**

| Store Name | Product_ Department | Count_No._of_ Sales transaction |
|---|---|---|
| Store No. 19 | Grocery | 3,595.00 |
| Store No. 19 | Household | 596 |
| Store No. 2 | Grocery | 3,441.00 |
| Store No. 2 | Household | 599 |
| Store No. 20 | Grocery | 3,518.00 |
| Store No. 20 | Household | 632 |
| Store No. 3 | Grocery | 3,508.00 |
| Store No. 3 | Household | 562 |
| Store No. 4 | Grocery | 3,394.00 |
| Store No. 4 | Household | 628 |
| Store No. 5 | Grocery | 3,416.00 |
| Store No. 5 | Household | 641 |
| Store No. 6 | Grocery | 3,604.00 |
| Store No. 6 | Household | 650 |
| Store No. 7 | Grocery | 3,524.00 |
| Store No. 7 | Household | 632 |
| Store No. 8 | Grocery | 3,436.00 |
| Store No. 8 | Household | 595 |
| Store No. 9 | Grocery | 3,466.00 |
| Store No. 9 | Household | 608 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

**Schema Version No. 5**

**Sum of dollar sales for product of all categories which were sold in New York**

**OOMDB SEMAN**

| Year | Product_ Category | Store_city | Sum_of_ dollar_sales |
|---|---|---|---|
| 2001 | Drinks | New York | 1,830,000.00 |
| 2001 | Food | New York | 4,545,000.00 |
| 2001 | Supplies | New York | 1,099,000.00 |
| 2002 | Drinks | New York | 1,994,000.00 |
| 2002 | Food | New York | 4,431,000.00 |
| 2002 | Supplies | New York | 1,140,000.00 |
| 2003 | Drinks | New York | 1,864,000.00 |
| 2003 | Food | New York | 4,486,000.00 |
| 2003 | Supplies | New York | 1,068,000.00 |

**MS. Analysis Manager**

| Year | Product_ Category | Store_city | Sum_of_ dollar_sales |
|---|---|---|---|
| 2001 | Drinks | New York | 1,830,000.00 |
| 2001 | Food | New York | 4,545,000.00 |
| 2001 | Supplies | New York | 1,099,000.00 |
| 2002 | Drinks | New York | 1,994,000.00 |
| 2002 | Food | New York | 4,431,000.00 |
| 2002 | Supplies | New York | 1,140,000.00 |
| 2003 | Drinks | New York | 1,864,000.00 |
| 2003 | Food | New York | 4,486,000.00 |
| 2003 | Supplies | New York | 1,068,000.00 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

Count customer numbers who bought product “Lasagna” or “Beef Stew” during Nov 10- 20, 2001

**OOMDB SEMAN**

| Year | Month | Product_Full Description | Count_ No_of_ Customers |
|---|---|---|---|
| 2001 | November | Beef Stew 13.5 oz | 42 |
| 2001 | November | Beef Stew 6 oz | 44 |
| 2001 | November | Beef Stew 9 oz | 38 |
| 2001 | November | Lasagna 13.5 oz | 37 |
| 2001 | November | Lasagna 6 oz | 42 |
| 2001 | November | Lasagna 9 oz | 34 |

**MS. Analysis Manager**

| Year | Month | Product_Full Description | Count_ No_of_ Customers |
|---|---|---|---|
| 2001 | November | Beef Stew 13.5 oz | 42 |
| 2001 | November | Beef Stew 6 oz | 44 |
| 2001 | November | Beef Stew 9 oz | 38 |
| 2001 | November | Lasagna 13.5 oz | 37 |
| 2001 | November | Lasagna 6 oz | 42 |
| 2001 | November | Lasagna 9 oz | 34 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

**Schema Version No. 6**

**Average dollar_sales by product subcategory and month in January to June**

**OOMDB SEMAN**

| Year | Month | Average_of_ Dollar_sales |
|---|---|---|
| 2001 | January | 5,538.46 |
| 2001 | February | 5,384.84 |
| 2001 | March | 5,528.76 |
| 2001 | April | 5,537.71 |
| 2001 | May | 5,471.37 |
| 2001 | June | 5,462.12 |
| 2002 | January | 5,496.87 |
| 2002 | February | 5,453.87 |
| 2002 | March | 5,496.47 |
| 2002 | April | 5,462.20 |
| 2002 | May | 5,439.64 |
| 2002 | June | 5,579.16 |
| 2003 | January | 5,420.88 |
| 2003 | February | 5,576.94 |
| 2003 | March | 5,553.77 |
| 2003 | April | 5,460.68 |
| 2003 | May | 5,455.31 |
| 2003 | June | 5,520.71 |

**MS. Analysis Manager**

| Year | Month | Average_of_ Dollar_sales | Difference |
|---|---|---|---|
| 2001 | January | 5,538.46 | 0 |
| 2001 | February | 5,384.84 | 0 |
| 2001 | March | 5,528.76 | 0 |
| 2001 | April | 5,537.71 | 0 |
| 2001 | May | 5,471.37 | 0 |
| 2001 | June | 5,462.12 | 0 |
| 2002 | January | 5,496.87 | 0 |
| 2002 | February | 5,453.87 | 0 |
| 2002 | March | 5,496.47 | 0 |
| 2002 | April | 5,462.20 | 0 |
| 2002 | May | 5,439.64 | 0 |
| 2002 | June | 5,579.16 | 0 |
| 2003 | January | 5,420.88 | 0 |
| 2003 | February | 5,576.94 | 0 |
| 2003 | March | 5,553.77 | 0 |
| 2003 | April | 5,460.68 | 0 |
| 2003 | May | 5,455.31 | 0 |
| 2003 | June | 5,520.71 | 0 |

**Schema Version No. 7**

**Difference between sum of dollar sales and dollar cost by store, product and year for store No.1 and product “Buffalo Jerky”, “Dried Grits”**

**OOMDB SEMAN**

| Year | Store_ Name | Product_Full Description | Sum_of_ Dollar_ sales | Sum_of_ Dollar_ costs | Gross_ profit |
|---|---|---|---|---|---|
| 2001 | Store No. 1 | Buffalo Jerky 3 oz | 95,000 | 66,500 | 28,500 |
| 2001 | Store No. 1 | Buffalo Jerky 4.5 oz | 134,000 | 93,800 | 40,200 |
| 2001 | Store No. 1 | Buffalo Jerky 6.7 oz | 127,000 | 88,900 | 38,1000 |
| 2001 | Store No. 1 | Dried Grits 2 oz | 98,000 | 68,600 | 29,400 |
| 2001 | Store No. 1 | Dried Grits 3 oz | 148,000 | 103,600 | 44,400 |
| 2001 | Store No. 1 | Dried Grits 4.5 oz | 119,000 | 83,300 | 35,700 |
| 2002 | Store No. 1 | Buffalo Jerky 3 oz | 139,000 | 97,300 | 41,700 |
| 2002 | Store No. 1 | Buffalo Jerky 4.5 oz | 96,000 | 67,200 | 28,800 |
| 2002 | Store No. 1 | Buffalo Jerky 6.7 oz | 95,000 | 66,500 | 28,500 |
| 2002 | Store No. 1 | Dried Grits 2 oz | 199,000 | 139,300 | 59,700 |
| 2002 | Store No. 1 | Dried Grits 3 oz | 95,000 | 66,500 | 28,500 |
| 2002 | Store No. 1 | Dried Grits 4.5 oz | 148,000 | 103,600 | 44,400 |
| 2003 | Store No. 1 | Buffalo Jerky 3 oz | 163,000 | 114,100 | 48,900 |

**MS. Analysis Manager**

| Year | Store_ Name | Product_Full Description | Sum_of_ Dollar_ sales | Sum_of_ Dollar_ costs | Gross_ profit |
|---|---|---|---|---|---|
| 2001 | Store No. 1 | Buffalo Jerky 3 oz | 95,000 | 66,500 | 28,500 |
| 2001 | Store No. 1 | Buffalo Jerky 4.5 oz | 134,000 | 93,800 | 40,200 |
| 2001 | Store No. 1 | Buffalo Jerky 6.7 oz | 127,000 | 88,900 | 38,1000 |
| 2001 | Store No. 1 | Dried Grits 2 oz | 98,000 | 68,600 | 29,400 |
| 2001 | Store No. 1 | Dried Grits 3 oz | 148,000 | 103,600 | 44,400 |
| 2001 | Store No. 1 | Dried Grits 4.5 oz | 119,000 | 83,300 | 35,700 |
| 2002 | Store No. 1 | Buffalo Jerky 3 oz | 139,000 | 97,300 | 41,700 |
| 2002 | Store No. 1 | Buffalo Jerky 4.5 oz | 96,000 | 67,200 | 28,800 |
| 2002 | Store No. 1 | Buffalo Jerky 6.7 oz | 95,000 | 66,500 | 28,500 |
| 2002 | Store No. 1 | Dried Grits 2 oz | 199,000 | 139,300 | 59,700 |
| 2002 | Store No. 1 | Dried Grits 3 oz | 95,000 | 66,500 | 28,500 |
| 2002 | Store No. 1 | Dried Grits 4.5 oz | 148,000 | 103,600 | 44,400 |
| 2003 | Store No. 1 | Buffalo Jerky 3 oz | 163,000 | 114,100 | 48,900 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

**OOMDB SEMAN**

| Year | Store_ Name | Product_Full Description | Sum_of_ Dollar_ sales | Sum_of_ Dollar_ costs | Gross_ profit |
|---|---|---|---|---|---|
| 2003 | Store No. 1 | Buffalo Jerky 4.5 oz | 137,000 | 95,900 | 41,100 |
| 2003 | Store No. 1 | Buffalo Jerky 6.7 oz | 129,000 | 90,300 | 38,700 |
| 2003 | Store No. 1 | Dried Grits 2 oz | 99,000 | 69,300 | 29,700 |
| 2003 | Store No. 1 | Dried Grits 3 oz | 112,000 | 78,400 | 33,600 |
| 2003 | Store No. 1 | Dried Grits 4.5 oz | 107,000 | 74,900 | 32,100 |

**MS. Analysis Manager**

| Year | Store_ Name | Product_Full Description | Sum_of_ Dollar_ sales | Sum_of_ Dollar_ costs | Gross_ profit |
|---|---|---|---|---|---|
| 2003 | Store No. 1 | Buffalo Jerky 4.5 oz | 137,000 | 95,900 | 41,100 |
| 2003 | Store No. 1 | Buffalo Jerky 6.7 oz | 129,000 | 90,300 | 38,700 |
| 2003 | Store No. 1 | Dried Grits 2 oz | 99,000 | 69,300 | 29,700 |
| 2003 | Store No. 1 | Dried Grits 3 oz | 112,000 | 78,400 | 33,600 |
| 2003 | Store No. 1 | Dried Grits 4.5 oz | 107,000 | 74,900 | 32,100 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

**Schema Version No. 8**

**Rank the unit sales by store state**

**OOMDB SEMAN**

| Store_state | Sum_of_ Unit_sales |
|---|---|
| PA | 46,475.00 |
| CA | 44,567.00 |
| MA | 23,271.00 |
| FL | 22,908.00 |
| CO | 22,864.00 |
| GA | 22,743.00 |
| TN | 22,590.00 |
| OH | 22,543.00 |
| MN | 22,542.00 |
| WA | 22,483.00 |
| NY | 22,457.00 |
| KY | 22,452.00 |
| AZ | 22,326.00 |
| MO | 22,261.00 |
| IL | 22,156.00 |
| TX | 22,110.00 |
| LA | 21,888.00 |
| DC | 21,812.00 |

**MS. Analysis Manager**

| Store_state | Sum_of_ Unit_sales |
|---|---|
| PA | 46,475.00 |
| CA | 44,567.00 |
| MA | 23,271.00 |
| FL | 22,908.00 |
| CO | 22,864.00 |
| GA | 22,743.00 |
| TN | 22,590.00 |
| OH | 22,543.00 |
| MN | 22,542.00 |
| WA | 22,483.00 |
| NY | 22,457.00 |
| KY | 22,452.00 |
| AZ | 22,326.00 |
| MO | 22,261.00 |
| IL | 22,156.00 |
| TX | 22,110.00 |
| LA | 21,888.00 |
| DC | 21,812.00 |

| Difference |
|---|
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

**Count number of customers influenced by promotion type “Big Promo” or “Blue Ribbon Discounts”**

**OOMDB SEMAN**

| Promotion_ Name | Count_No._ Of_customers |
|---|---|
| Big Promo | 7,576.00 |
| Blue Ribbon Discounts | 7,522.00 |

**MS. Analysis Manager**

| Promotion_ Name | Count_No._ Of_customers |
|---|---|
| Big Promo | 7,576.00 |
| Blue Ribbon Discounts | 7,522.00 |

| Difference |
|---|
| 0 |
| 0 |

# CHAPTER 7

# DISCUSSION AND CONCLUSION

## 7.1 Discussion and Conclusion

We have presented our proposed solution in detail on how to apply the Object-Oriented Multidimensional Modeling as well as the Object-oriented versioning approach to MD schema evolution support. In addition, the proposed functions and algorithms were also formally introduced. Hereafter, we discussed our solution by presenting the implementation of OOMDB SEMAN as a demonstrable prototype in chapter 5 and 6.

The findings indicate that the OO modeling enables the design for MDB in a more flexible and feasible way. In the OO context, the MDB schema comprises dimension class and fact class, which are easy to analyze and maintain. Furthermore, the schema versioning approach serves as a major mechanism to support the change of OOMDB schema in a very appropriate way. Since not only the change of schema is supported but also the historical schema version and its instance are well maintained. Concerning the primary concept of versioning approach, any evolved schema and its instances are not considered to be useless, but they are well kept and can actively respond to the queries specified by the user in a proper timeline. Additionally, the two proposed algorithms: *Forward Schema Version Compatibility Algorithm* and *Backward Schema Version Compatibility Algorithm*, are proven to be powerful to support the entire process of MDB schema evolution in terms of the versioning concept. In addition, the MDB schema evolution operations and the effect of MDB schema changes are presented in terms of the descriptive algebra appropriately.

Finally, we developed the prototype system called OOMDB SEMAN (Object-Oriented Multidimensional DataBase Schema Evolution MANager) to serve as an easy-to-use tool for the support of MDB schema update. In the experiment, the base schema is derived from the real-world grocery system and up to 1,000,000 records of data were used for the test. To verify our proposed solution, the Analysis Manager, which is a system module of Microsoft SQL Server 2000, is used as a standard tool to validate the correctness and consistency of the functionality against the OOMDB SEMAN. Actually, the verification of the correctness and consistency is regarded in two levels: structural schema level and instance level. For the verification of structural schema transformation, 100 schema versions are built as the test case. The resulting change of each schema version is compared to the system log archive and algorithm trace log to find out whether the evolution tasks of the presented case and prototype are identical or not. Therefore, we can check the result of the schema version change in a case-by-case fashion. To verify the correctness of OOMDB SEMAN functionality in the instance level, we set up 50 MD queries to perform the tests upon the 100 schema versions. The tests were applied to our prototype and the MS. Analysis Manager. Then the result of all test cases among these two systems was compared.

The result of the overall validations revealed that the OOMDB SEMAN was functionally correct and consistent in all versions. This also implies that our proposed algorithms are practical and useful for supporting the evolution process of the MDB schema.

In summary, the major benefits derived from the OOMDB SEMAN are:

1. OOMDB SEMAN facilitates the friendly interface that allows the schema modification to be performed via an OOMDB graphical conceptual model. The tool is thus responsible for performing the necessary steps in a consistent and semantically correct way of the OOMDB schema evolution. Accordingly, the schema designer does not need to adjust configurations of metadata of the front-end tool or database schema.

2. Apart from the function of the schema evolution control, OOMDB SEMAN also incorporates the prominent features of typical OLAP tools. The user thus can perform the analytical functions consecutively.

3. Based on the versioning technique, the tool supports multi-users to perform the analytical queries via the different schema versions.

## 7.2 Future works

For future research, we point out several interesting issues as follows:

1. First of all, in order to increase the performance, a common strategy of the physical design issue and optimization of ROLAP is a promising way to apply to OOMDB SEMAN. Particularly, indexing techniques such as Bitmap and B-tree can be a practical indexing scheme for adoption. In addition, the clustering technique is also an interesting research area for enhancing the capacity of the multidimensional database system. Also, the analysis of the impact of schema evolution on the clustered database is deemed prominent.

2. Temporal Versioning Model is another plausible model to support the full-fledged schema evolution since using the temporal versioning supports not only the version history tracking but also the multidimensional data from different versions which can be compared and mapped to the static structure. OOMDB SEMAN might employ this concept to promote the capability of the data analysis according to different schema versions and the complete plan of semantic change of the schema version in terms of temporal reflection.

3. Finally, the design of transaction management of the multidimensional information system is also an interesting issue since the transaction control is an important requirement for the concurrent MD schema changes. Consequently,

multidimensional time stamp ordering or multidimensional schema locking techniques are worth further studying.

# REFERENCES

1. Agrawal, R., Gupta, A., Sarawagi, S., "Modeling Multidimensional Databases". In 13th International Conference On Data Engineering, (ICDE'97) pages 232-243, Birmingham, U.K., April, 1997.
2. Cabbibo, L. Torlone, R., "Querying Multidimensional databases". In 6th International Workshop on Database Programming Languages (DBPL 1997).
3. M. Gyssens and L.V.S. Lakshmanan, "A Foundation for Multi-dimensional Databases", In 33rd International Conference On Very Large Databases, pages 105/115,1997.
4. T. B. Nguyen, A. M. Tjoa, and R. R. Wagner, "An Object Oriented Multidimensional Data Model for OLAP", Proceeding of 1st International Conference on Web-Age Information Management (WAIM), volume 1846 of LNCS, pages 83-94, Springer, 2000.
5. Lehner, W., "Modelling Large Scale OLAP Scenarios". Lecture Notes in Computer Science, number 1337 in Proceedings of the 6th International Conference On Extending Database Technology, (EDBT'97), pages 153-167, 1998.
6. Chaudhuri S, Dayal U, "An overview of data warehousing and OLAP Technology", VLDB Conference, 1996.
7. Trujillo, J., Palomar, M. " Object Oriented Approach to Multidimensional Database", Conceptual Modeling (OOMD)" DOLAP, 1998.
8. M. Blaschka: "FIESTA: A Framework for Schema Evolution in Multidimensional Information Systems" Proceedings of 6th CASE Doctoral Consortium, 1999, Heidelberg, Germany.
9. John Joseph, Satish Thatte, Craig W. Thomphon, and David L. Wells. Report on the

Object-Oriented Database Workshop, Held in Conjunction with OOPSLA’ 88. SIGMOD Record, 18(3): 78-101, 1989.

10. Li, X. and Tari, Z. “Class Versioning for the Schema Evolution”, Proceedings of the Australian Database Conference (ADC), Perth, February 1998.
11. A. Gupta. I. S. Mumick “Maintenance of Materialized Views: Problems, Techniques, and Applications”, Data Engineering Bulletin, June 1995.
12. Simon Monk and Ian Sommerville, Schema Evolution in OODBs Using Class Versioning. Sigmod Record, Vol. 22, No.3, 1993.
13. Seven-Eric Lautemann, “An Introduction to Schema Versioning in OODBMS”, Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA), 1996.
14. Sven-Eric Lautemann, “Schema Versions in Object-Oriented Database Systems”, Proceedings of the 5th International Conference on Database Systems for Advanced Applications, 1997.
15. C. Hurtado, A.O, Mendelzon, A.A. Vaisman, “Maintaining Data Cubes under Dimension Updates”, Proceedings of the ICDE’99, Sydney Australia, 1999.
16. C. Hurtado, A.O, Mendelzon, A.A. Vaisman, "Updating OLAP Dimensions", Proceedings of the 2nd International Workshop on Data Warehousing and OLAP, Kansas City, Missouri, USA, November 1999.
17. J. Eder, C. Koncilia, “Changes of Dimension Data in Temporal Data Warehouses” In proceedings of the DaWak 2001 Conference, Munich, Germany, 2001.
18. P. Vassiliadis and T.K. Sellis, “A Survey of Logical Models of OLAP Databases,” ACM SIGMOD Record, Vol.28, n. 4, 1999, pp. 64-69.
19. J.L. Mitpanont, S. Chirawattanakij, “Multidimensional Object Schema : An Object – Oriented Approach in a Multidimensional Database Design”, Proceedings of the fifth National Computer Science and Engineering Conference( NCSEC 2001) Thailand, 2001.
20. S. Samtani, M. K. Monania, V. Kumar, Y. Kambayashi, “Recent Advances and

Research Problems in Data Warehousing", Advances in Database Technologies, ER '98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work support and Spatio-Temporal Data Management, Singapore, November 1998.

21. Buzydlowski, J. W., song, II-Y., Hassell, L, "A Framework for Object-Oriented On-Line Analytic Processing", DOLAP 1998.

22. John F. Roddick, "A Survey of Schema Versioning Issues for Database Systems", Information and Software Technology, Vol. 37 No.7, pp. 383-393, 1995.

23. M. Mohania, S. Konomi and Y. Kambayashi, "Incremental Maintenance of Materialized Views", Proceedings of 8th International Conference on database and Expert Systems Applications (DEXA '97). Springer-Verlag, 1997.

24. Li, C., Wang, X.S, "A Data Model for Supporting On-Line Analytical Processing", CIKM 1996.

25. Vassiliadis, P., "Modeling Multidimensional Databases, Cubes and Cube operations", Proceedings of 10th Scientific and Statistical Database Management conference (SSDBM'98), Capri, Italy, June 1998.

26. R. Bliujute, S. Saltenis, G. Slivinskas, C.S. Jensen,"Systematic Change Managment in Dimensional DataWarehousing", Proceedings of the 3rd International BalticWorkshop on DB and IS, 1998.

27. P. Chamoni and S. Stock, "Temporal Structures in Data warehousing", Proceedings of the DaWaK'99 Conference, Florence, Italy, 1999.

28. Panel on schema evolution and version management (1989) Report on the object-oriented database workshop, SIGMOD Record, 18(3), 1989. pp. 78-101.

29. R. Kimball, "The Data Warehouse Toolkit", Wiley, New York, 1996.

30. M. Mohania, G. Dong, "Algorithms for Adapting Materialized Views in Data Warehouses", International Symposium on Cooperative Database Systems for Advanced Applications, Kyoto, Japan, World-Scientific, 1996.

31. M. Mohania, "Avoiding Re-computation: View Adaptation in Data Warehouses", Proceedings of 8th International Database Workshop Hong-Kong, Springer LNCS, 1997.

32. Z. Bellahsene, "View Adaptation in Data Warehousing Systems", Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA),Vienna, Austria, August 1998.

33. Banerjee, J., C. Hong-Tai, J. F. Garza, W. Kim, D. Woelk, N. Ballou and K. Hyoung-Joo, “Data Model Issues for Object-Oriented Applications”, ACM Trans. On Office Information Systems, 5(1), pp 3-26, 1987.

34. Banerjee, J., W. Kim, H. Kim and H. Korth, “Semantics and Implications of Schema Evolution in Object-Oriented Databases.”, Proceedings of ACM SIGMOD, San Francisco. 1987.

35. Body M., Miquel M., Be’dard Y., Tchounikine A,“ A Multidimensional Structure and Multiversion for OLAP Applications”, DOLAP 2002, ACM Fifth International Workshop on Data Warehousing and OLAP, November 8, 2002, McLean, VA, Proceedings. ACM 2002.

36. B. Staudt Lerner and A. N. Habermann, “Beyond Schema Evolution to Database Reorganization”, In N. Meyrowitz, editor, Proceedings of the 5th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)ACM, Vol.25, No.10, October, 1990.

37. J.L. Mitpanont, C. Puapanniwat, “An Evaluation of the Hierarchical Join Index in ROLAP”, Proceedings of the sixth National Computer Science and Engineering Conference( NCSEC 2002) Thailand, 2002.

# APPENDIX

# APPENDIX A

## A.1 Taxonomy of Schema Changes in an Object-oriented Database System

In [33,34], Banerjee et al, defined the taxonomy for schema changes which is generally applicable for OODB. The OODB schema changes are categorized into two groups: changes of class definitions and changes of class lattice. Figure A1 shows the taxonomy of changes of class definition.

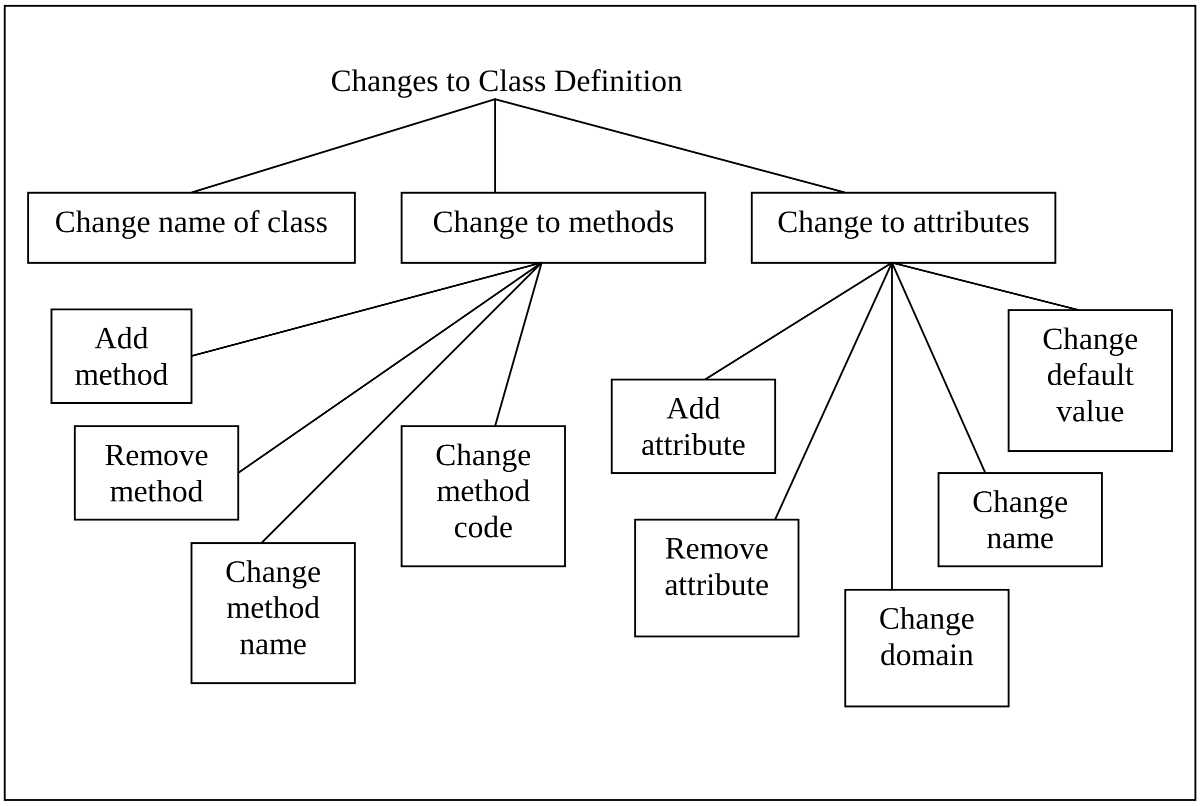


Figure A1: Taxonomy of Changes to Class Definition

Figure A2 displays the taxonomy of changes of class lattice.

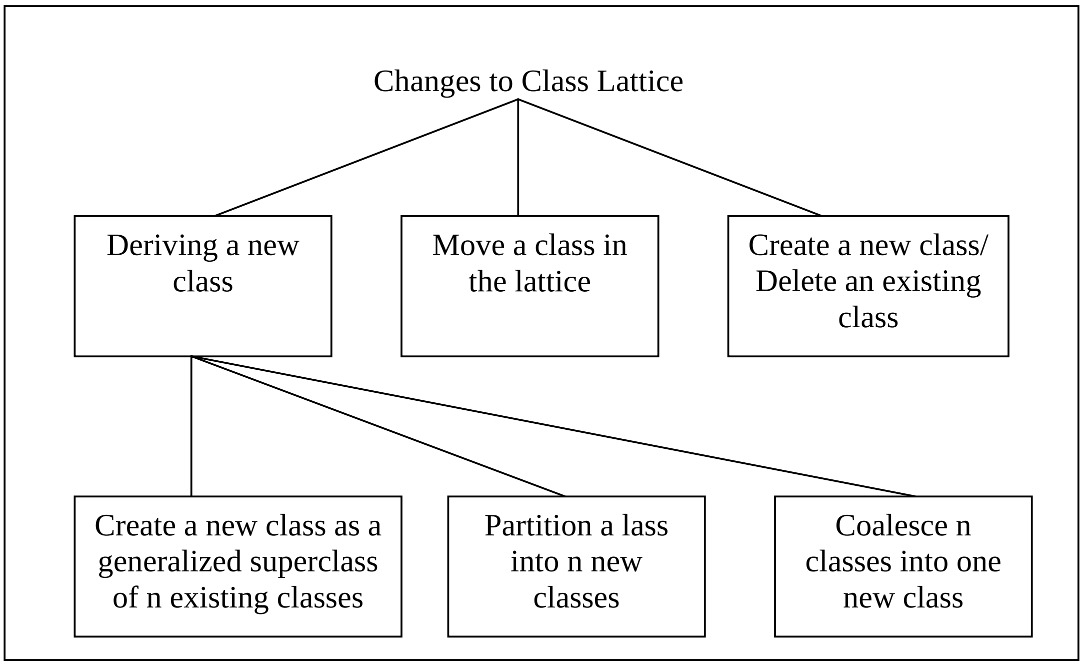

Changes to Class Lattice
Deriving a new class
Move a class in the lattice
Create a new class/ Delete an existing class
Create a new class as a generalized superclass of n existing classes
Partition a lass into n new classes
Coalesce n classes into one new class

Figure A2: Taxonomy of Changes to Class Lattice

## A.2 Taxonomy of Schema Changes in Multidimensional Structure

In this section, three prominent approaches of taxonomy of MDB schema changes are presented : Approach of Body, Miquel, Be'dard and Tchounikine [35], Approach of Hurtado, Mendelzon, and. Vaisman [15,16], and Approach of Blaschka [8].

### 1. Approach of Body, Miquel, Be'dard and Tchounikine

1. Dimension Schema Evolution
    - Creation and deletion of a dimension
    - Creation and deletion of a hierarchy
    - Creation and deletion of a level
    - Move of a level in the hierarchical schema structure

2. Evolution on Members: Simple Operations
    - Creation of a member
    - Deletion of a member
    - Transformation of a member (change of an attribute, its name or meaning)
    - Merging of n members into one member
    - Splitting of one member into n members
    - Reclassification of a member in the dimension structure

3. Evolution on Members: Complex Operations
    - Decreasing: splitting and deletion
    - Increasing: creation and merging
    - Partial annexation: splitting and merging

**2. Approach of Hurtado, Mendelzon, and. Vaisman**

1. Primitive Operators
    - Generalize: Adds a new level above a pre-existing one
    - Specialize: Adds a new level below the current bottom level
    - Relate: Adds a new edge, between two parallel levels
    - Unrelate: Deletes an edge between two levels.
    - Delete Level: Deletes a level with the precondition that the new hierarchy must have a unique bottom.
    - Add Instance: Adds a value to the domain of some rollup function.
    - Delete Instance: Delete a value from a domain of level *l*.

2. Complex Operators (instances update operators)
    - Reclassify: Reclassifies the dimension domain
    - Split: Splits the existing level to n levels and assign the value of instances of the new generated level
    - Merge: Merges two instances of a dimension into a single one.
    - Update: Updates the changes of instances value cause by any modifications

**3. Approach of Blaschka: FIESTA**

1. Modification of a dimension level
    - Insert level
    - Delete level

2. Modification of an attribute
    - Insert attribute
    - Delete attribute
    - Connect attribute to dimension level
    - Disconnect attribute from dimension level
    - Connect attribute to fact

- Disconnect attribute from fact

3. Modification of a classification relationship
    - Insert Classification relationship
    - Delete Classification relationship

4. Modification of a fact
    - Insert fact
    - Delete fact
    - Insert dimension level into fact
    - Delete dimension level from fact

# BIOGRAPHY

| | |
|---|---|
| **NAME** | Mr. Somchart Fugkeaw |
| **DATE OF BIRTH** | 27 September 1979 |
| **PLACE OF BIRTH** | Bangkok, Thailand |
| **INSTITUTIONS ATTENDED** | Thammasat University, 1996 - 2000 : Bachelor of Business Administration (Management Information System) |
| | Mahidol University, 2001 - 2004 : Master of Science (Computer Science) |
| **POSITION&OFFICE** | 2001-2003, Thai Digital ID Co., Ltd. Position: Certification Authority Operator |
| | 2004-Present, Thai Digital ID Co., Ltd. Position: Acting CA Operation Manager |