

MORM: A FACT-ORIENTED CONCEPTUAL MODELING APPROACH  
TO DESIGNING  
DATA WAREHOUSES AND OLAP APPLICATIONS

by

ROBERT R. PAYNE

B.Comm. (Co-op), Memorial University of Newfoundland, 1994

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Faculty of Commerce and Business Administration)

We accept this thesis as conforming  
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October 2003

© Robert R. Payne, 2003

## Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Robert Payne

Name of Author *(please print)*

10/12/03

Date

Title of Thesis: MORM: A Fact-Oriented Conceptual Modeling Approach  
to Designing Data Warehouses and OLAP Applications

Degree: MSCB

Year: 2004

## **ABSTRACT**

The interest in data warehouses and OLAP applications in recent years is largely due to the promise of improved decision-making made possible by integrating data from numerous different sources. The underlying data structures required to support the analytical requirements of these systems clearly justifies the need for a distinct data modeling approach, particularly at the conceptual level. As a direct response to the inability of conventional data modeling methods to represent multidimensional semantics, multidimensional modeling has emerged and brought with it a variety of new multidimensional concepts. Several models have been proposed to represent these concepts, but a complete and natural approach does not exist that adequately conceptualizes and communicates multidimensional designs to business and technical users alike. To address the fundamental deficiencies and shortcomings of existing models, we propose a fact-oriented approach to conceptual multidimensional modeling. Our approach is a specialization of Object Role Modeling in which we define additional constructs and guidelines to consider multidimensional properties at the conceptual level. We believe our utilization of the fact-oriented paradigm provides us with a conceptual multidimensional model that is more natural and simpler than existing models. To demonstrate its practicality, we apply our approach to a case study and demonstrate how our model can be implemented using existing technologies.

# TABLE OF CONTENTS

ABSTRACT.....	ii
TABLE OF CONTENTS.....	iii
LIST OF TABLES .....	vi
LIST OF FIGURES .....	vii
ACKNOWLEDGEMENTS.....	ix
1. INTRODUCTION .....	1
1.1. MOTIVATION.....	2
1.2. OBJECTIVES .....	3
1.3. OUTLINE .....	4
2. INTRODUCTION TO DATA WAREHOUSING .....	5
2.1. INTRODUCTION .....	5
2.2. DATA WAREHOUSING OVERVIEW.....	5
2.3. LOGICAL ARCHITECTURE.....	6
2.3.1. <i>Data Source Layer</i> .....	6
2.3.2. <i>Data Storage Layer</i> .....	7
2.3.3. <i>Application Layer</i> .....	7
2.3.4. <i>Presentation Layer</i> .....	8
2.4. PHYSICAL ARCHITECTURE .....	8
2.4.1. <i>Relational OLAP Architecture</i> .....	8
2.4.2. <i>Multidimensional OLAP Architecture</i> .....	9
2.5. FUNDAMENTAL DATA WAREHOUSE PROCESSES.....	10
2.5.1. <i>Data Staging Services</i> .....	11
2.5.2. <i>Query Services</i> .....	12
2.6. SUMMARY.....	13
3. MODELING THE DATA WAREHOUSE .....	14
3.1. INTRODUCTION .....	14
3.2. INFORMATION LEVELS .....	14
3.2.1. <i>Conceptual Model</i> .....	16
3.2.2. <i>Logical Model</i> .....	17
3.2.3. <i>Physical Model</i> .....	18
3.3. CONVENTIONAL DATA MODELING APPROACHES .....	19
3.3.1. <i>Entity-Relationship Modeling</i> .....	19
3.3.1.1. Barker Notation.....	20
3.3.1.2. Information Engineering (IE) Notation.....	21
3.3.1.3. Integrated Definition (IDEF1X) Notation.....	21
3.3.2. <i>Object-Oriented Modeling</i> .....	22
3.3.3. <i>Fact-Oriented Modeling</i> .....	23
3.3.4. <i>Conventional Data Modeling and Data Warehouses</i> .....	25
3.4. MULTIDIMENSIONAL DATA MODELING .....	28
3.4.1. <i>Multidimensional Concepts Through an Example</i> .....	30



3.4.2. <i>Business Processes</i> .....	31
3.4.3. <i>Events</i> .....	31
3.4.3.1. <i>Derived Measures</i> .....	31
3.4.3.2. <i>Additivity</i> .....	32
3.4.4. <i>Dimensions</i> .....	33
3.4.4.1. <i>Classification Hierarchies</i> .....	33
3.4.4.2. <i>Strictness &amp; Completeness</i> .....	35
3.4.4.3. <i>Categorization of Dimensions</i> .....	36
3.4.4.4. <i>Many-to-Many Relationships Between Events and Dimensions</i> .....	36
3.4.4.5. <i>Degenerate Dimensions</i> .....	37
3.5. <i>RELATED MULTIDIMENSIONAL DATA MODELING WORK</i> .....	37
3.5.1. <i>Conceptual Level</i> .....	38
3.5.1.1. <i>Multidimensional Entity Relationship Model (M/ER)</i> .....	38
3.5.1.2. <i>Star Entity Relationship Model (starER)</i> .....	39
3.5.1.3. <i>Dimensional Fact Model (DFM)</i> .....	40
3.5.1.4. <i>GOLD Model</i> .....	41
3.5.2. <i>Logical Level</i> .....	42
3.5.3. <i>Physical Level</i> .....	43
3.5.4. <i>Formal Level</i> .....	43
3.5.5. <i>Shortcomings of Existing Models</i> .....	44
3.6. <i>SUMMARY</i> .....	46
4. <i>FACT-ORIENTED MULTIDIMENSIONAL MODELING</i> .....	47
4.1. <i>INTRODUCTION</i> .....	47
4.2. <i>KEY DESIGN CONSIDERATIONS</i> .....	47
4.3. <i>WHY USE OBJECT ROLE MODELING?</i> .....	48
4.3.1. <i>Advantages of Using ORM</i> .....	48
4.3.1.1. <i>Conceptual Modeling Evaluation Criteria</i> .....	49
4.3.2. <i>Disadvantages of Using ORM</i> .....	50
4.4. <i>MULTIDIMENSIONAL OBJECT ROLE MODELING (MORM)</i> .....	51
4.4.1. <i>Business Processes</i> .....	54
4.4.1.1. <i>Event &amp; Dimension Constructs</i> .....	56
4.4.1.2. <i>Families of Business Processes</i> .....	58
4.4.2. <i>Events</i> .....	59
4.4.2.1. <i>Atomic Measures</i> .....	60
4.4.2.2. <i>Derived Measures</i> .....	61
4.4.2.3. <i>Additivity</i> .....	62
4.4.3. <i>Dimensions</i> .....	62
4.4.3.1. <i>Classification Hierarchies</i> .....	63
4.4.3.2. <i>Strictness</i> .....	67
4.4.3.3. <i>Completeness</i> .....	68
4.4.3.4. <i>Categorization of Dimensions</i> .....	69
4.4.3.5. <i>Many-to-Many Relationships Between Events and Dimensions</i> .....	70
4.4.3.6. <i>Degenerate Dimensions</i> .....	71
4.5. <i>MORM DESIGN GUIDELINES</i> .....	72
4.5.1. <i>MORM Level 0: Preliminary Segmentation</i> .....	72
4.5.2. <i>MORM Level 1: Business Process Family Definition</i> .....	73
4.5.3. <i>MORM Level 2: Business Process Definition</i> .....	75
4.5.4. <i>MORM Level 3: Event Definition</i> .....	76

4.5.5. <i>MORM Level 4: Dimension Definition</i> .....	77
4.5.6. <i>Design Guideline Summary</i> .....	78
4.6. AN EVALUATION OF MORM.....	79
4.7. SUMMARY.....	81
5. APPLYING MORM: A CASE STUDY.....	82
5.1. INTRODUCTION.....	82
5.2. DEVELOPMENT TOOLS.....	82
5.2.1. <i>Conceptual &amp; Logical Modeling Tool: VisioModeler™</i> .....	83
5.2.2. <i>Relational Database: Microsoft® SQL Server™ 2000</i> .....	84
5.2.3. <i>OLAP Tool: Microsoft® SQL Server™ Analysis Services</i> .....	85
5.3. STEP 1: CREATING THE CONCEPTUAL MORM SCHEMA.....	85
5.3.1. <i>VisioModeler Diagram Workspace</i> .....	86
5.3.2. <i>Creating a MORM Project</i> .....	88
5.3.3. <i>Creating MORM Schemas</i> .....	90
5.4. STEP 2: MAPPING THE LOGICAL SCHEMA.....	91
5.4.1. <i>Building the Data Dictionary</i> .....	92
5.4.2. <i>Relational Mapping (Rmap) Procedure</i> .....	93
5.4.3. <i>Editing the Logical Model</i> .....	95
5.5. STEP 3: GENERATING A PHYSICAL SCHEMA.....	95
5.5.1. <i>Schema Generation Options</i> .....	96
5.5.2. <i>Generating Directly Through ODBC</i> .....	96
5.5.3. <i>Generating a DDL Script</i> .....	99
5.6. STEP 4: BUILDING AN OLAP CUBE.....	99
5.6.1. <i>Setting up the Database &amp; Data Source</i> .....	100
5.6.2. <i>Building the Cube</i> .....	101
5.6.3. <i>Designing Storage and Processing the Cube</i> .....	102
5.6.4. <i>Browsing Cube Data</i> .....	103
5.7. AN EVALUATION OF OUR CASE STUDY.....	104
5.7.1. <i>Hierarchies: Multiple, Alternative Path, and Shared</i> .....	105
5.7.2. <i>Non-Strictness</i> .....	106
5.7.3. <i>Many-to-Many Relationships Between Events and Dimensions</i> .....	107
5.8. SUMMARY.....	109
6. CONCLUSIONS & FUTURE RESEARCH.....	110
6.1. THESIS SUMMARY.....	110
6.2. CONTRIBUTIONS.....	112
6.3. LIMITATIONS AND FUTURE RESEARCH.....	114
BIBLIOGRAPHY.....	116
APPENDIX A: ORM CONSTRUCTS.....	120
APPENDIX B: MORM SCHEMA FOR RETAIL CASE STUDY.....	122
APPENDIX C: MORM IMPLEMENTATION RESULTS.....	127

## **LIST OF TABLES**

TABLE 3-1: OPERATIONAL DATABASE AND DATA WAREHOUSE DIFFERENCES .....	26
TABLE 3-2: COMPARISON OF EXISTING CONCEPTUAL MULTIDIMENSIONAL MODELS .....	45
TABLE 4-1: MORM CONSTRUCTS AND ASSOCIATED DESCRIPTIONS .....	53
TABLE 4-2: CONCEPTUAL SCHEMA DESIGN PROCEDURE (CSDP) .....	53
TABLE 4-3: RETAIL POINT OF SALE FACTS .....	54
TABLE 4-4: FACTS WITH REFERENCE MODES OMITTED .....	55
TABLE 4-5: FACT TYPES WITH VALUES OMITTED .....	55
TABLE 4-6: SALES TRANSACTION EVENT FACT TYPES .....	60
TABLE 4-7: PRODUCT DIMENSION FACT TYPES .....	63
TABLE 4-8: MORM LEVEL 0 DESIGN GUIDELINES .....	73
TABLE 4-9: FOUR LEVELS OF A MORM SCHEMA .....	73
TABLE 4-10: MORM LEVEL 1 DESIGN GUIDELINES .....	73
TABLE 4-11: MORM LEVEL 2 DESIGN GUIDELINES .....	75
TABLE 4-12: MORM LEVEL 3 DESIGN GUIDELINES .....	76
TABLE 4-13: MORM LEVEL 4 DESIGN GUIDELINES .....	77
TABLE 4-14: MORM DESIGN GUIDELINE SUMMARY .....	79
TABLE 5-1: MULTIDIMENSIONAL MODELING IMPLEMENTATION LIFECYCLE .....	82
TABLE 5-2: MORM DESIGN GUIDELINE #0B .....	90
TABLE A-1: ORM CONSTRUCTS AND ASSOCIATED DESCRIPTIONS .....	121
TABLE C-1: DETAILS OF MORM IMPLEMENTATION RESULTS .....	129

## LIST OF FIGURES

FIGURE 2-1: LOGICAL DATA WAREHOUSE ARCHITECTURE .....	6
FIGURE 2-2: RELATIONAL OLAP ARCHITECTURE LAYERS .....	9
FIGURE 2-3: MULTIDIMENSIONAL OLAP ARCHITECTURE LAYERS .....	10
FIGURE 2-4: BASIC DATA WAREHOUSE PROCESSES .....	11
FIGURE 3-1: SAMPLE CONCEPTUAL DATA MODEL .....	17
FIGURE 3-2: SAMPLE LOGICAL DATA MODEL .....	18
FIGURE 3-3: SAMPLE PHYSICAL DATA MODEL .....	19
FIGURE 3-4: CLASSIC NOTATION FOR ER MODELING.....	20
FIGURE 3-5: BARKER NOTATION FOR ER MODELING.....	21
FIGURE 3-6: INFORMATION ENGINEERING (IE) NOTATION FOR ER MODELING.....	21
FIGURE 3-7: IDEF1X NOTATION FOR ER MODELING .....	22
FIGURE 3-8: UNIFIED MODELING LANGUAGE CLASS DIAGRAM .....	23
FIGURE 3-9: OBJECT ROLE MODELING SCHEMA.....	25
FIGURE 3-10: REPRESENTATION OF THE CUBE METAPHOR.....	29
FIGURE 3-11: EXAMPLES OF CLASSIFICATION HIERARCHIES .....	34
FIGURE 3-12: SAMPLE M/ER MODEL FOR VEHICLE REPAIR .....	39
FIGURE 3-13: SAMPLE STARER MODEL FOR MORTGAGE REPAYMENT .....	39
FIGURE 3-14: SAMPLE DIMENSIONAL FACT MODEL FOR INVENTORY MANAGEMENT .....	40
FIGURE 3-15: SAMPLE GOLD MODEL FOR RETAIL SALES .....	41
FIGURE 3-16: LOGICAL STAR SCHEMA FOR RELATIONAL DATABASES.....	42
FIGURE 4-1: GRAPHICAL NOTATION FOR NEW MORM CONSTRUCTS .....	52
FIGURE 4-2: SCHEMA FOR POS RETAIL SALES BUSINESS PROCESS.....	56
FIGURE 4-3: SCHEMA FOR RETAIL BUSINESS PROCESS FAMILY .....	59
FIGURE 4-4: SCHEMA FOR SALES TRANSACTION EVENT.....	60
FIGURE 4-5: DERIVATION RULE FOR THE PROFIT MEASURE .....	61
FIGURE 4-6: MULTIPLE CLASSIFICATION HIERARCHIES IN MORM .....	64
FIGURE 4-7: ALTERNATIVE PATH HIERARCHIES IN MORM .....	65
FIGURE 4-8: SHARED HIERARCHIES IN MORM .....	66
FIGURE 4-9: STRICTNESS & NON-STRICTNESS IN MORM.....	67
FIGURE 4-10: COMPLETENESS IN MORM.....	68
FIGURE 4-11: CATEGORIZATION OF DIMENSIONS IN MORM.....	69
FIGURE 4-12: MANY-TO-MANY RELATIONSHIP BETWEEN EVENT & DIMENSION .....	71
FIGURE 4-13: MORM LEVEL 1 - RETAIL BUSINESS PROCESS FAMILY.....	74
FIGURE 4-14: MORM LEVEL 2 - POS RETAIL SALES BUSINESS PROCESS.....	75
FIGURE 4-15: MORM LEVEL 3 - SALES TRANSACTION EVENT.....	76
FIGURE 4-16: MORM LEVEL 4 - STORE DIMENSION .....	78
FIGURE 5-1: MORM MODEL AND ASSOCIATED DICTIONARY DOCUMENT.....	87
FIGURE 5-2: VISIOMODELER TOOL AND CONSTRAINT PALETTES.....	87
FIGURE 5-3: VISIOMODELER FACT EDITOR WINDOW.....	88
FIGURE 5-4: FACT EDITOR CONSTRAINTS AND ASSOCIATED DATA EXAMPLES.....	88
FIGURE 5-5: VISIOMODELER'S PROJECT WINDOW .....	89
FIGURE 5-6: OUTPUT WINDOW SHOWING BUILD RESULTS.....	92
FIGURE 5-7: LOGICAL MODEL MAPPED FROM ORM SCHEMA .....	94
FIGURE 5-8: OPTIONS WITHIN VISIOMODELER'S GENERATE WIZARD .....	96

FIGURE 5-9: ASSOCIATING AN ODBC DRIVER WITH A VISIOMODELER DRIVER .....	97
FIGURE 5-10: ODBC DATA SOURCE DEFINITION .....	98
FIGURE 5-11: TABLE PREVIEW IN VISIOMODELER GENERATE WIZARD .....	98
FIGURE 5-12: DDL SCRIPT GENERATED BY VISIOMODELER .....	99
FIGURE 5-13: ANALYSIS SERVICES DATABASE OBJECT .....	100
FIGURE 5-14: ANALYSIS SERVICES CUBE EDITOR .....	101
FIGURE 5-15: CUBE PROCESSING USING THE DESIGN WIZARD .....	102
FIGURE 5-16: FINAL CUBE PROCESSING RESULTS .....	102
FIGURE 5-17: FILTERING EXAMPLE WITHIN THE CUBE BROWSER .....	103
FIGURE 5-18: DRILL DOWN EXAMPLE WITHIN THE CUBE BROWSER .....	103
FIGURE 5-19: MODELING MANY-TO-MANY RELATIONSHIPS .....	108
FIGURE A-1: GRAPHICAL NOTATION OF ORM CONSTRUCTS .....	120
FIGURE B-1: MORM LEVEL 1 - RETAIL BUSINESS PROCESS FAMILY .....	122
FIGURE B-2: MORM LEVEL 2 - POS RETAIL SALES BUSINESS PROCESS .....	122
FIGURE B-3: MORM LEVEL 2 - INVENTORY BUSINESS PROCESS .....	123
FIGURE B-4: MORM LEVEL 3 - SALES TRANSACTION EVENT .....	123
FIGURE B-5: MORM LEVEL 3 - INVENTORY EVENT .....	124
FIGURE B-6: MORM LEVEL 4 - STORE DIMENSION .....	124
FIGURE B-7: MORM LEVEL 4 - CUSTOMER DIMENSION .....	125
FIGURE B-8: MORM LEVEL 4 - PRODUCT DIMENSION .....	125
FIGURE B-9: MORM LEVEL 4 - TIME DIMENSION .....	126

## **ACKNOWLEDGEMENTS**

I owe thanks for contributions to this thesis to a great many people. First of all, I would like to thank my thesis Supervisor, Dr. Carson Woo, for his commitment, support, and invaluable guidance throughout this project. I am also indebted to my other committee members, Dr. Yair Wand and Dr. Jacob Steif, for their insightful advice in revising this work and their thought-provoking questions during my thesis defense.

I am also grateful to Angus Livingstone at UBC's University Industry Liaison Agency (UILO) for sponsoring the early data analysis work that introduced me to data warehousing and eventually led to this thesis. My research was also advanced by spending several wonderful months in Chicago, Illinois as part of an exceptional consulting group, Fathom Solutions. Specifically, I thank Brad at Fathom for being a source of great discussion and always reminding me of the value of practical work.

Last, but certainly not least, I would like to thank my family for their never-ending support. I would especially like to thank Jill for knowing how and when to steer my mind both towards and away from research. A source of constant inspiration and encouragement, Jill has been incredibly supportive and tolerant of my obsession to achieve this goal.

# **1. INTRODUCTION**

Recent years have witnessed the dramatic evolution and acceptance of a new type of management information system known as the data warehouse. The interest in this decision support technology is largely due to the promise of improved decision-making and planning made possible by gaining efficient access to data from numerous different information sources. As originally defined by Inmon (1996), a data warehouse is “a subject-oriented, integrated, non-volatile, time variant collection of data in support of management’s decisions” (p. 33).

The data warehouse is often the underlying database that supports an integrated data architecture to deliver decision oriented data structures to on-line analytical processing (OLAP) applications. In this role, data warehouses contrast operational databases that support daily operations and on-line transaction processing (OLTP).

Wu and Buchmann (1997) present significant differences between data warehouse and operational systems to justify the need for separate underlying databases. Chaudhuri and Dayal (1997), also suggest differences in functionality and performance requirements as valid reasons for differentiating the design and development of data warehouses from that of conventional operational systems. These viewpoints suggest that the decision-support focus of data warehouses demands data model design aligned with user perspectives and the analytical processing to be performed rather than application specific business needs. The point has also been argued by Boehnlein and Ulbriche-vom Ende (1999) that both the static and dynamic influences of analytical requirements and their underlying data sources clearly illustrate the need for a distinct comprehensive data warehouse modeling approach.

## **1.1. Motivation**

While conventional modeling techniques are well proven for transaction processing systems, their deficiencies in modeling data warehouses have been well documented (Golfarelli, Maio, & Rizzi, 1998a; Kimball, 1997; Raden, 1995). The main reason for the deficiency in these arguments is the underlying normalization premise which provides an efficient means to store data but does not satisfy analytical and decision support requirements. As argued by Kimball, Reeves, Ross and Thornthwaite (1998), these models should not be used as the basis for data warehouses because the atomic detail of normalized models often confuses users and cannot be easily navigated by analysis tools.

As a direct response to the inability of conventional conceptual modeling methods to represent multidimensional semantics, the multidimensional view of data (cube or hypercube) arose (Chaudhuri & Dayal, 1997). This view introduced a variety of new modeling concepts, including facts and dimensions, additivity, derived measures, classification hierarchies, and the categorization of dimensions.

Despite the growth of data warehousing and OLAP applications, existing multidimensional modeling methods do not adequately capture the inherent semantics and a commonly accepted standard does not exist to indicate what should be represented in a multidimensional scheme (Abello, Samos, & Saltor, 2001). While it is widely recognized that data warehouses are based on the logical star schema, there is no standard conceptual data modeling technique for data warehousing and OLAP applications. Consequently, user analysis via a common framework is difficult and there is no consistent basis for solving conceptual multidimensional modeling problems with an intuitive and complete conceptual model.



Several works have been proposed that provide significant support for multidimensional constructs, but these approaches are far removed from natural language and difficult to populate with example information, making it challenging to conceptualize and validate multidimensional designs. As a result, a complete and natural design technique does not exist that adequately conceptualizes and communicates multidimensional data designs to both business and technical users.

## **1.2. Objectives**

To address the fundamental deficiencies and shortcomings of existing multidimensional models, we propose a fact-oriented approach to multidimensional modeling. We define our approach as a specialization of Object Role Modeling (ORM) by defining additional graphical constructs and guidelines that consider key multidimensional properties at the conceptual level. We use ORM as a basis for our fact-oriented approach because we believe it considers an information system's structural properties at the conceptual level more naturally than existing multidimensional models or other conventional approaches.

The practical goals of our fact-oriented modeling approach are to help designers capture and satisfy complex modeling requirements, help end users better understand the structure and navigation paths of the data warehouse, and facilitate communication between business users and data modelers. In support of our goals, the main objectives of this thesis are summarized as follows:

- 1. to provide a natural, simple, and expressive approach to modeling multidimensional data at the conceptual level;*

2. *to present a set of design guidelines that provide data modelers with a systematic approach to building conceptual multidimensional models using our method; and*
3. *to test our approach by attempting to solve a practical data analysis problem through a case study implemented using existing technologies.*

### **1.3. Outline**

To accomplish our objectives, we have organized the remainder of the thesis as follows. Chapter 2 provides an overview of data warehousing and OLAP applications. Various architectural components are discussed, including logical layers, physical layers, and basic warehouse processes. Chapter 3 examines data modeling, highlighting the differences between traditional applications and data warehouses. Multidimensional concepts are presented through an example and existing multidimensional modeling works are reviewed. Chapter 4 introduces our conceptual multidimensional modeling approach by providing constructs and design guidelines for its use. Chapter 5 presents a case study demonstrating how our model can be implemented using existing technologies. Details are presented for conceptual model development, logical and physical schema mapping, and OLAP cube generation. We also report on our experiences gained in using the model. Finally, chapter 6 summarizes findings and contributions of the thesis and proposes future research directions.

## **2. INTRODUCTION TO DATA WAREHOUSING**

### **2.1. Introduction**

This chapter provides an overview of Data Warehousing and OLAP Applications. We discuss the logical architecture of a data warehouse through a presentation of its basic information service and communication layers. These include data source, data storage, application, and presentation layers. After presenting the logical view we outline physical architectures that may be mapped onto the logical architecture. Categorized by the approaches taken by desktop tools to implement data access, these physical architectures include Multidimensional OLAP (MOLAP) and Relational OLAP (ROLAP). We conclude the chapter with a discussion of the basic procedures within a data warehouse as grouped into two categories - data staging services and query services.

### **2.2. Data Warehousing Overview**

According to the Data Warehousing Institute (2000), the data warehousing industry encompasses a host of disciplines and technologies used to analyze information, including data modeling, data migration and transformation, data quality, business intelligence, data marts, on-line analytical processing, database management, data mining, and knowledge discovery. All of these terms can be classified into a broad category of information analysis known as decision support, which is one of the primary uses of data warehouses.

The importance of data warehousing in the commercial segment appears to be due to a need for enterprises to gather information from transaction processing systems into a single place for in-depth analysis (Widom, 1995). To gather this information in a typical data warehouse, information from a variety of sources is extracted, transformed, and

cleansed, and business rules are enforced to help clarify and standardize the data to ensure consistency. The following sections present the logical architecture required to support these and other warehouse processes and outline the physical architectures that may be mapped onto the logical architecture.

## 2.3. Logical Architecture

Figure 2-1 presents a typical logical data warehouse architecture that extends that of Wu and Buchmann (1997) and Kimball et al. (1998). This architecture contains the basic information service and communication layers of the data warehouse. These layers are discussed in the sections that follow.

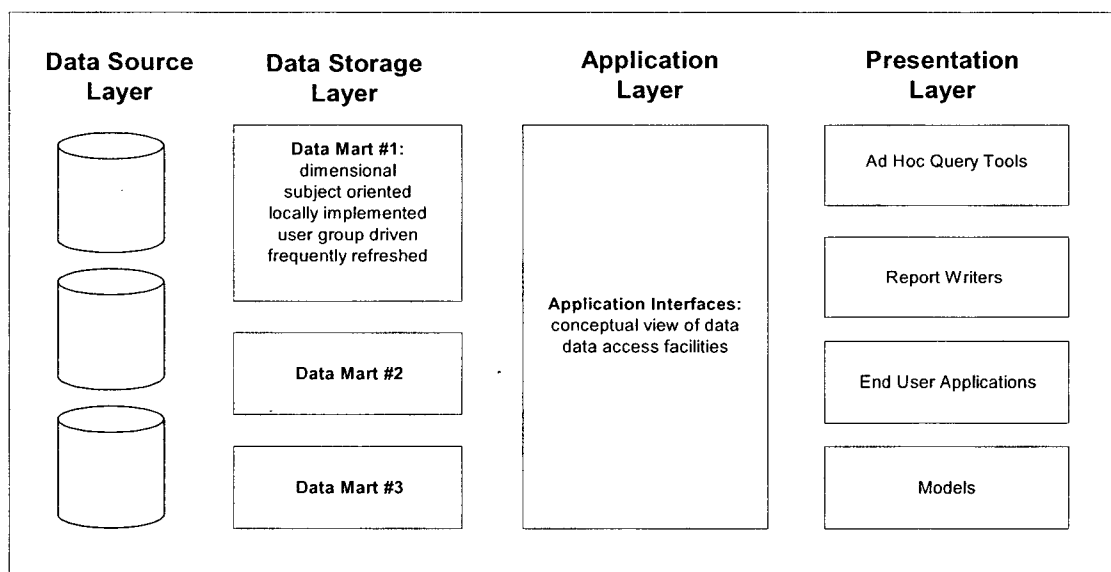


Figure 2-1: Logical Data Warehouse Architecture

### 2.3.1. Data Source Layer

Source systems capture business transactions and include operational systems and databases. These operational systems are often referred to as online transaction processing systems and are optimized for storing and updating large volumes of data gathered one record at a time. The purpose of these systems is known in advance

whereas it is not known for data warehouses. Examples of such systems are order entry, invoicing, inventory, and general ledger systems. Source systems should be thought of as outside the warehouse and may also include data from sources external to a company (e.g. marketing research data). The net result of these OLTP systems is the production of large volumes of data but the data gathered and stored is not always easily accessible to end users. This highlights the fact that the intent of these systems is not analytical processing.

### **2.3.2. Data Storage Layer**

The data storage layer provides services for the efficient storage, retrieval, and management of large amounts of data. It usually refers to the data warehouse database, which is frequently updated on a controlled basis using extract, transform, and load (ETL) routines on source system data and is the union of all its constituent data marts. In this context, Kimball et al. (1998) refer to a data mart as “the subset of all the data or a restriction of the data warehouse relevant to a specific business process or group” (p. 18) while a data warehouse usually serves the entire enterprise.

### **2.3.3. Application Layer**

The application layer provides services to conceptually arrange data in the format requested by end user applications. It provides data access facilities suitable for specific applications, including data model transformation between conceptual and logical schemas. This layer may also contain utilities to generate extracts frequently offloaded to desktop resident OLAP tools. Services for the arrangement of this data are application dependant. As Wu and Buchmann (1997) state, this is advantageous because if the requirements of the applications change, only the application layer needs to change.

#### **2.3.4. Presentation Layer**

To complete the logical architecture and deliver data from transactional systems to end users who make strategic and tactical decisions, a presentation layer is needed. The decision support systems that comprise this layer range from simple query tools to sophisticated data mining and modeling applications that find trends in the data.

Most often the presentation layer consists of an OLAP application. These graphical presentation and reporting systems allow users to intuitively, quickly, and flexibly manipulate operational data using familiar business terms in order to provide analytical insight. Many of these tools provide value added information through techniques such as exception highlighting, trend analysis, and statistics development.

### **2.4. Physical Architecture**

The following sections present physical architectures that may be mapped onto the logical architecture. These architectures are categorized by desktop tool approaches to implementing data access functionality and include multidimensional OLAP (MOLAP) and relational OLAP (ROLAP). The main premise of MOLAP is that data must be stored multidimensionally to be viewed multidimensionally while the ROLAP premise is that OLAP capabilities are best provided directly against a relational database.

#### **2.4.1. Relational OLAP Architecture**

As illustrated in Figure 2-2, the three-tier ROLAP architecture leverages relational databases to provide multidimensional analysis. The database layer typically utilizes relational databases for data storage, access, and retrieval processes. The application logic layer is the ROLAP engine, which executes multidimensional reports

from multiple end users. The ROLAP engine integrates with a variety of presentation layers, through which users perform OLAP analysis.

ROLAP servers sit between a relational database server and a client front-end tool. These servers extend traditional database servers with special middleware to efficiently support OLAP queries and analysis. These queries are then evaluated in terms of views that are identified beforehand and used to generate SQL.

As suggested by Microstrategy (1995), the main strength of ROLAP tools is that they exploit the scalability and transactional features of relational databases while the major drawback is the performance bottleneck that can result from performing OLAP style querying and generating SQL.

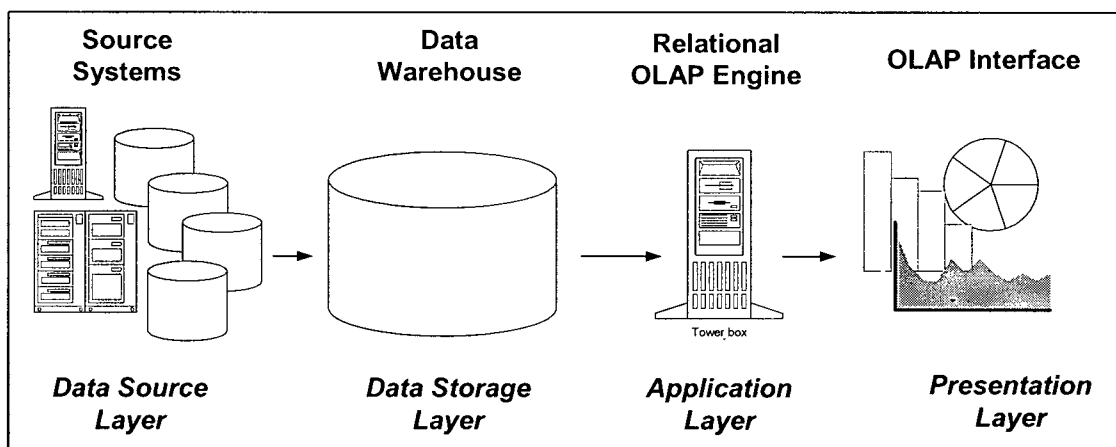
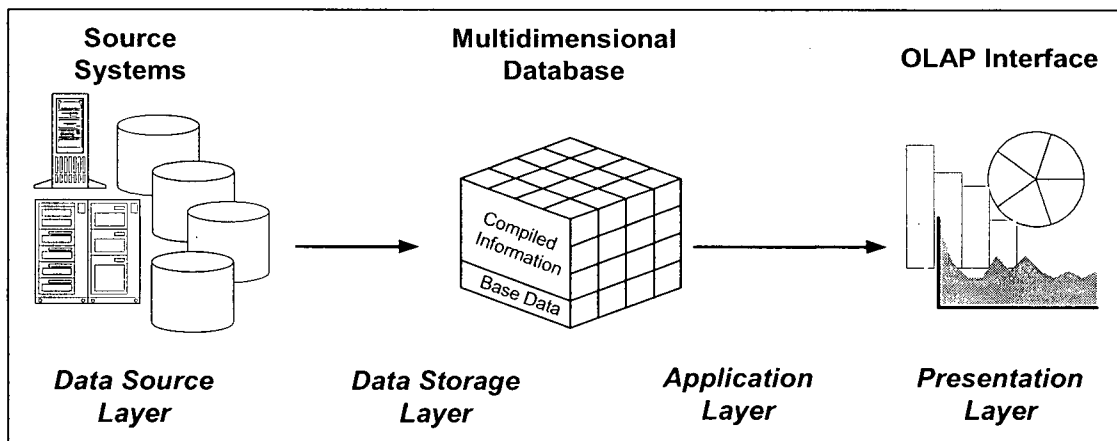


Figure 2-2: Relational OLAP Architecture Layers

#### 2.4.2. Multidimensional OLAP Architecture

Illustrated in Figure 2-3, MOLAP is a two-tier, client/server architecture in which a proprietary multidimensional database (MDB) serves as both the storage layer and the application layer. In the storage layer, the MDB system handles data storage, access, and retrieval functions. These databases contain n-dimensional arrays where each dimension has an associated hierarchy of levels of consolidated data. Data is loaded into an MDB

via batch routines and calculations are performed to aggregate along dimensions and fill the MDB's array structures. The application layer executes OLAP requests and integrates with the presentation layer to provide an interface through which end users view and request OLAP analysis.



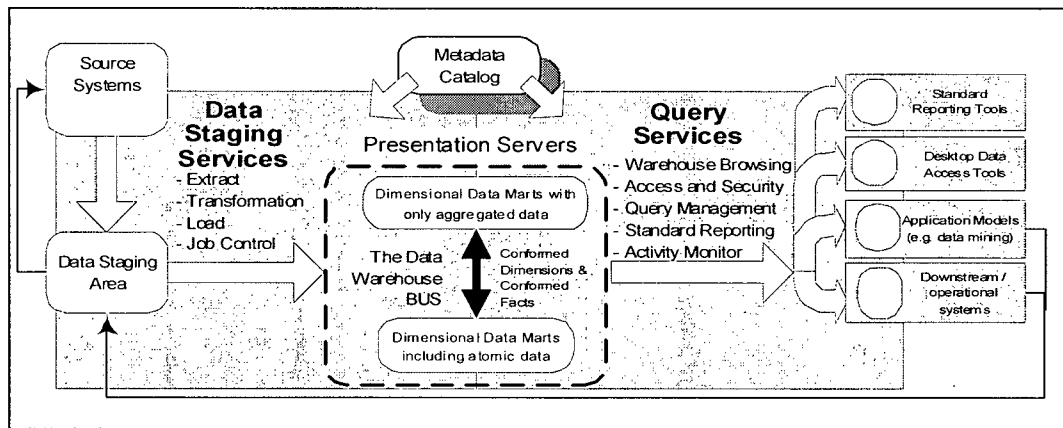
**Figure 2-3: Multidimensional OLAP Architecture Layers**

The significant advantage of the MOLAP approach is faster performance resulting from the indexing properties of the proprietary multidimensional storage structures while the major drawback is poor storage utilization, especially when the data set is sparse (Microstrategy, 1995).

## **2.5. Fundamental Data Warehouse Processes**

Figure 2-4 depicts the basic processes of a data warehouse as described by Kimball et al. (1998). They group these processes into two broad service categories – data staging services and query services.





**Figure 2-4: Basic Data Warehouse Processes**

Adapted from *The Data Warehouse Lifecycle Toolkit* (p. 329), by R. Kimball, L. Reeves, M. Ross, & W. Thornthwaite, 1998, New York: John Wiley & Sons.

### 2.5.1. Data Staging Services

One of the most significant tasks in building the data warehouse is moving data from numerous legacy systems into the data warehouse. At a high level, data moves from the source systems to a staging area using data staging services. These services are driven by metadata that describe data sources, targets, dependencies, etc. Often termed ETL services, they extract data from the data sources, transform and integrate the data, then load it into the warehouse. The timing of ETL processes is dependant on the characteristics of the source data, and may vary in frequency to be executed daily, weekly, monthly, etc. Common tools used for these processes include products such as Ascential DataStage, SAS ETL, and Microsoft DTS.

To effectively use ETL services, the entire data staging process is managed through job control services. These services define a series of steps, specify relations and dependencies among these steps, and capture metadata regarding their progress and statistics. These services can be implemented as SQL stored procedures or in an advanced tool designed to help manage the processes. Exception and error handling processes are

usually implemented as well to catch referential integrity violations and handle other unrecoverable errors.

### **2.5.2. Query Services**

Query services allow a user to formulate a query, execute it against the database, and respond to the request with a result set. Only presentation servers provide query services as they store and present data in a multidimensional format. As shown in Figure 2-4, the flow of data from the presentation server to the end user is supported by metadata from the metadata catalog.

OLAP browsing services help users navigate the warehouse by using some form of browsing tool. This tool is usually linked to a catalog containing business definitions and data elements for particular subject areas. Popular services and operations supported by OLAP tools include pivoting, roll-up, drill-down, and slice-and-dice. These services refer to the various manipulations that can be performed on query result sets.

Although complex analysis techniques have gained considerable attention with the advent of OLAP systems, managed query environments still exist that use standard reporting tools. These queries can come in the form of ad-hoc report requests or standard, fixed format reports. Oftentimes, queries that begin as ad-hoc requests become standard reports.

Other services found in the warehouse include activity monitoring and security services. Activity monitoring captures information about system performance and usage statistics to help with marketing and capacity planning. Security services facilitate database connections and rely on authentication and authorization processes to identify users and determine access rights.

## **2.6. Summary**

This chapter has provided an overview of data warehousing and OLAP applications. We introduced the logical architecture, which consists of data source, data storage, application, and presentation layers. These layers include operational systems and databases that capture business transactions, components for storing, retrieving, and managing data, components to conceptually arrange data in end user applications, and finally services to deliver data to the user. We also illustrated how the physical architecture of a data warehouse can be mapped on its logical layers in several ways. In a MOLAP solution analytical data is extracted and stored in a multidimensional database, while the ROLAP approach provides analytical capabilities directly against a relational database. The chapter concluded with a discussion of the basic processes within a data warehouse as presented in two broad categories. Data staging services are focused on getting data into the data warehouse and include ETL services, job control, exception and error handling processes. The second category, query services, focuses on getting data out of the warehouse and includes browsing, reporting and query management.

## **3. MODELING THE DATA WAREHOUSE**

### **3.1. Introduction**

In this chapter we provide an overview of the theoretical and practical aspects of modeling data for traditional OLTP systems, data warehouses, and OLAP applications. We first examine the basic concepts of data modeling by distinguishing conceptual, logical, and physical design phases. We will look at conventional data modeling methods and present an overview of several approaches, including Entity Relationship Modeling, Object-Oriented Modeling, and Fact-Oriented Modeling. After presenting conventional approaches, we will highlight differences in the data structures of traditional OLTP applications and decision support applications to distinguish multidimensional and conventional data modeling. To gain a better understanding of the structural properties of multidimensional data, we provide an overview of multidimensional concepts through an example. We conclude the chapter with a review of existing multidimensional models as discussed in the literature. A brief look is taken at physical and logical models but the emphasis is on several of the most popular conceptual models attempting to address multidimensional requirements.

### **3.2. Information Levels**

Databases are major productivity tools for information-oriented businesses, however, for a database to be used effectively its data should be well-designed, correct, and easy to access. Designing a database involves analyzing and representing data in a formal model of the application area an organization must understand for a particular system. The application area being modeled is typically part of the real world the modeler is interested in and has been referred to by Halpin (1995) as the universe of

discourse. A data modeling method allows business users and data architects to describe the universe of discourse clearly and precisely to achieve consensus on the definition of its contents. According to Halpin (1995), such a modeling method comprises both a language and a procedure to guide modelers in using the language to construct models. A language has associated syntax, semantics, and pragmatics and may be graphical and/or textual.

Data models make extensive use of graphical representations and natural language to visualize information needs of an application and gauge how completely and accurately data structures reflect an application domain. Several direct benefits of this visualization include improved communication between modeler and user, more understandable solutions, and early detection of missed requirements and modeling errors. Design improvements stemming from data modeling generally translate to fewer construction errors and less expense as inaccuracies do not filter through to later stages of software development and result in costly code changes. Once complete, data models serve as architectural blueprints for database and application development.

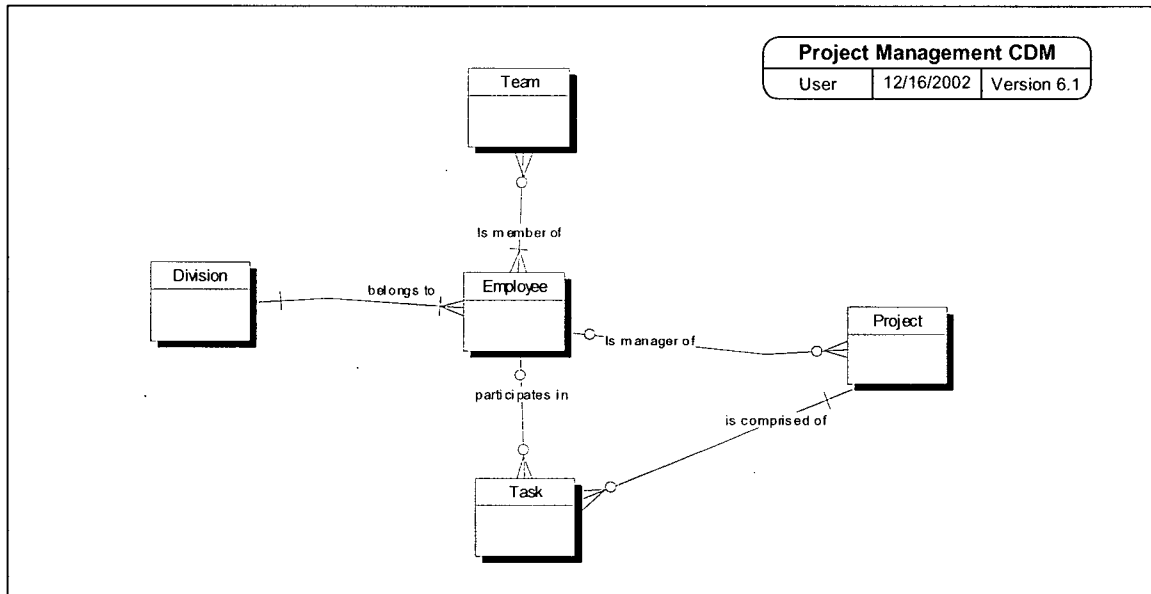
Most literature distinguishes between conceptual, logical, and physical database development phases when it comes to the subject of database design and development (Elmasri & Navathe, 1994; Batani, Ceri, & Navathe, 1992; Halpin 2001). The common theme in these works is that the terms conceptual, logical, and physical differentiate levels of abstraction in data models. These various layers are not defined by an accepted authority but are generally understood by data architects and modelers. The models may appear in different manners, some approaches dealing with only the physical and logical models, while others offer elements of all three. Combined, the conceptual, logical, and

physical models comprise a complete data model representing the highest level of design abstraction to the lowest level of implementation detail of a particular application.

Adopting this view of data modeling, we distinguish three different kinds of data models based on the constructs they provide and the database design phase they are associated with - Conceptual, Logical and Physical. Conceptual models are close to the way users perceive the data and are independent of any implementation. Secondly, logical models are understandable by end users but consider the underlying Database Management System (DBMS) used in the implementation. Finally, those at the physical level depend on the specific database used and describe the details of how data is actually stored in the computer. These levels are further described in the following sections.

### ***3.2.1. Conceptual Model***

The first step of database design is usually developing the conceptual data model of an application. Considered the highest level of database abstraction, conceptual design portrays the application domain using terms and concepts understandable to the user while ignoring logical and physical level aspects. The conceptual model is concerned with depicting data from the business users point of view, and thus, is said to represent an abstraction of the real world view and understanding of data (Batra, Hoffer, & Bostrom, 1990). The conceptual model suppresses non-critical details in order to emphasize business rules and user objects using concepts people are used to working with. As illustrated in the sample in Figure 3-1, conceptual models typically include only significant entities that have business meaning, along with their relationships and possibly a few significant attributes.

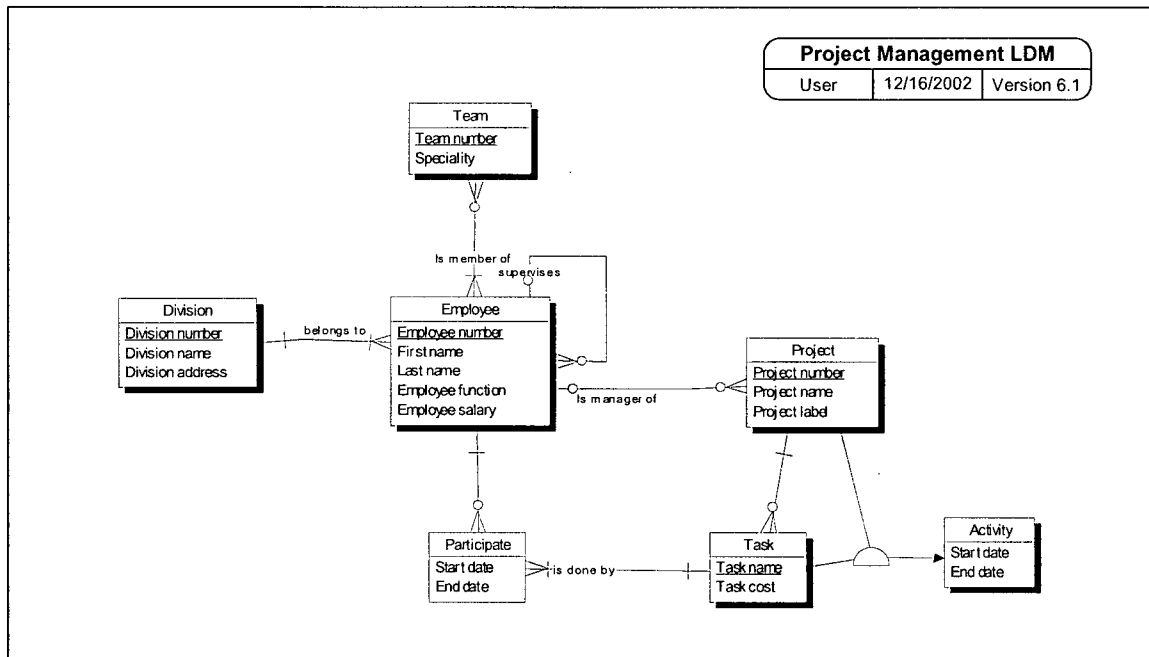


**Figure 3-1: Sample Conceptual Data Model**

According to Batra et al., (1990) the process of deriving and analyzing data inherent to a business situation and mapping the objects of this understanding of reality constitutes a discovery phase. This discovery phase consists of two parts - the first involves elicitation of the information requirements from users and the second involves conceptual representation of information requirements into a conceptual model.

### **3.2.2. Logical Model**

The logical database design phase typically follows conceptual design and converts the model into a lower-level structure for implementation purposes. To do this, an appropriate class of logical data model (e.g. relational, hierarchic, network) is chosen and a logical design is expressed in terms of the abstract database structures for that model. As shown in the relational model sample in Figure 3-2, information is stored in tables and constraints are expressed using primary and foreign key declarations.



**Figure 3-2: Sample Logical Data Model**

While various logical model classes exist, the predominance of relational databases has meant the majority of today's logical models are schemas conforming to relational theory. As introduced by Codd (1970), relational theory involves normalized relations where each data entry is atomic and stored in tables treated as mathematical relations. Relational schemas are most often in first normal form and do not include specific details for each relational DBMS implementation.

### **3.2.3. Physical Model**

The physical model specifies how the logical model will be instantiated in a particular DBMS product (e.g. Oracle, Sybase, etc.). As illustrated in the example in Figure 3-3, physical models include detailed table information specific to a particular product or version, as well as configuration choices for the database instance. Other details include physical storage options for index construction, key constraints, views and referential integrity maintenance.



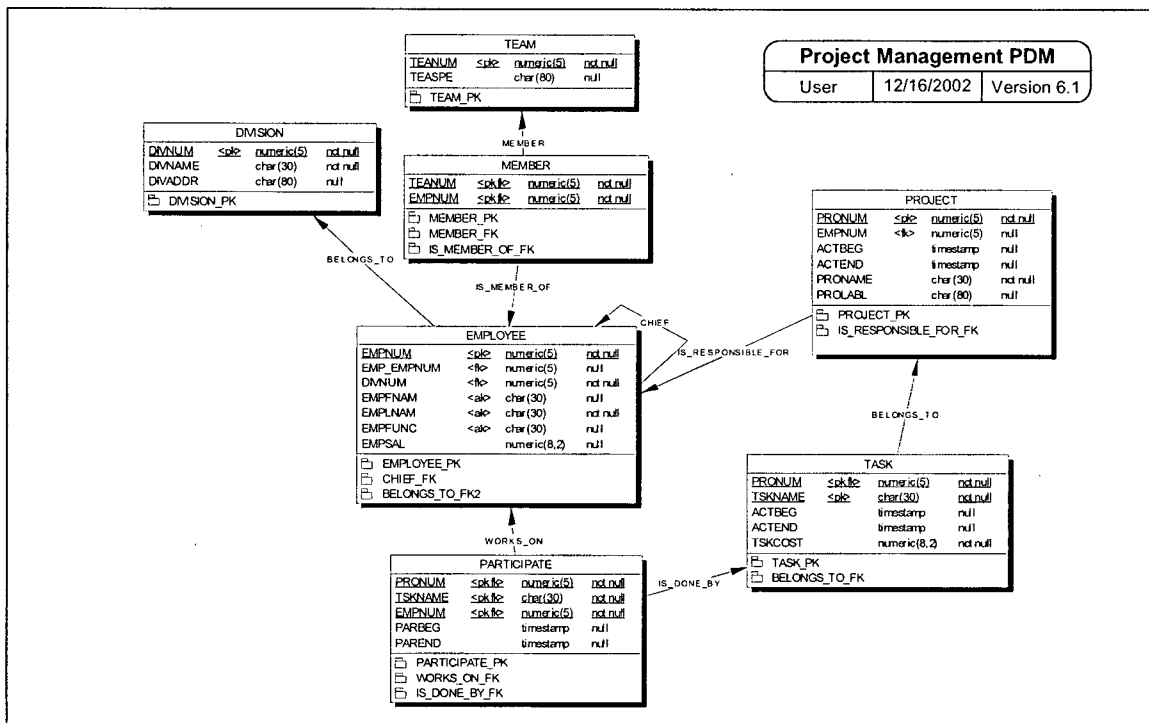


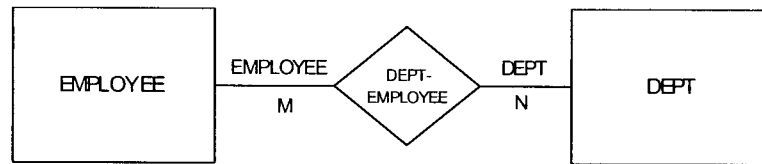
Figure 3-3: Sample Physical Data Model

### 3.3. Conventional Data Modeling Approaches

Many forms of symbolic notation have been developed to enable data models to represent various levels of abstraction. Some of these notations are lexical, others graphical, and others a combination of both. The following sections outline three of the most popular conventional notations for information modeling at the conceptual level.

#### 3.3.1. Entity-Relationship Modeling

The most common technique for conceptually modeling data in operational systems is the Entity Relationship (ER) model defined by Chen (1976). Shown in Figure 3-4, Chen's classic ER notation models entities that participate in relationships. For example, to model an employee working for a department, a relationship is assigned between the Employee and Department entities.

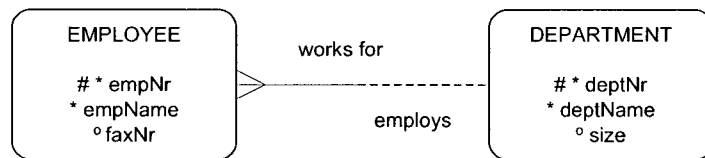


**Figure 3-4: Classic Notation for ER Modeling**

Introduced in the 1970s, Chen's model has evolved over time to incorporate extensions, variations, and improvements including the Extended Entity Relational Model (Teorey, Yang, & Fry, 1986) that captures detailed features of an information model. These different versions support different concepts and often use different symbols for the same concepts. A number of the extensions and variations have been incorporated in Computer Assisted Software Engineering (CASE) products employing the ER methodology. While there is no single standard ER notation, the most popular versions of ER are the Barker and Information Engineering (IE) notations. Although not a true ER representation, another popular notation is IDEF1X, which is a mixture of ER and relational approaches. We provide a brief overview of these three ER notations in the following sections.

#### **3.3.1.1. Barker Notation**

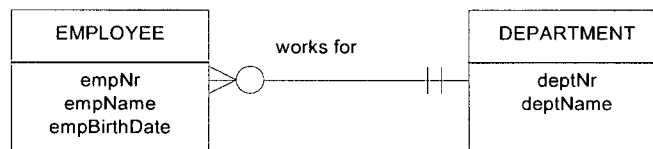
Originally proposed by Barker (1990), Oracle later adopted this notation in its CASE tools. Shown in Figure 3-5, the Barker notation represents entities as named, soft rectangles with a list of attributes. A hash (#) indicates the primary identifier of an entity. An asterisk (\*) or heavy dot (•) indicates an attribute is mandatory, while a superscript "O" (°) indicates it is optional. All relationships are binary and are shown as named lines. A solid half-line denotes a mandatory role and a dotted half-line denotes an optional role. A crow's foot indicates the cardinality "many" and its absence indicates "one".



**Figure 3-5: Barker Notation for ER Modeling**

### 3.3.1.2. Information Engineering (IE) Notation

Information Engineering was originated by Finkelstein (1989) and later adopted by Martin (1990). Today, different versions of IE exist and the style has become the basis for a number of CASE products, including Sybase's PowerDesigner. As Figure 3-6 illustrates, IE displays entity types as named rectangles with a list of attributes. Relationships are binary and denoted by named lines. A crow's foot indicates "many", a stroke indicates "one", two strokes indicate exactly one and a circle indicates optional.

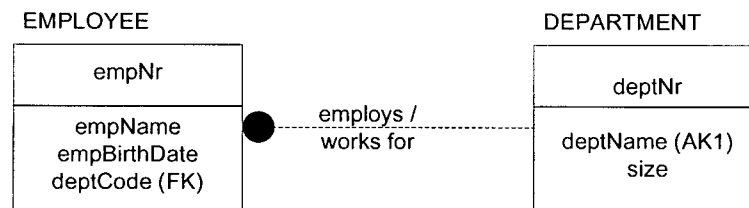


**Figure 3-6: Information Engineering (IE) Notation for ER Modeling**

### 3.3.1.3. Integrated Definition (IDEF1X) Notation

IDEF1X was developed in the late 1970's and later extended into a standard adopted by the U.S. Air Force as part of a required methodology for government projects. Originally, IDEF1X was a member of a family of Integrated Definition (IDEF) languages developed by the Air Force for Integrated Computer Aided Manufacturing (ICAM). Of the different languages defined for different tasks, IDEF1 was initially developed for conceptual data modeling and later "extended" as IDEF1X for logical data modeling. The current version (NIST, 1993) continues to be widely used for database design and has been adopted by many CASE tool vendors, including Computer Associates' ERwin.

IDEF1X is a hybrid modeling language, combining conceptual constructs (e.g. entity, relationship) with relational database constructs (e.g. foreign keys). IDEF1X models may be viewed at three levels – an ER view, a key-based view, and a fully attributed view. The ER view is used early in the design process and simply includes entity types and relationships with no attributes or identification schemes. The key based view includes at least all the key-based attributes and classifies relationships as identifying or non-identifying. Shown in Figure 3-7, the fully attributed view, as its title indicates, includes all attributes.



**Figure 3-7: IDEF1X Notation for ER Modeling**

Here, entity types are shown as named rectangles. Attributes are listed inside the rectangles with the primary key in the top compartment. Alternate keys are denoted with “(AKn)” and foreign keys with “(FK)”. Connection relationships are foreign key references from the child to parent and are shown with a dot “•” at the child end.

### **3.3.2. Object-Oriented Modeling**

Object-oriented modeling is an approach that encapsulates both data and behavior within objects. The most influential object-oriented approach that exists is the Unified Modeling Language (Booch, Rumbaugh, & Jacobson, 1999), which has been adopted by the OMG as a method for object oriented analysis and design. Though mainly focused on the design of object oriented programming code, UML can be used for modeling database

applications. UML is supported in various CASE tools, including Rational Rose, one of the most well known visual modeling tools for object-oriented modeling.

Of its nine diagrams, UML's Class diagram is used to specify static data structures by supplementing its predefined notations with user-specific notations. When stripped of implementation detail, Class diagrams are similar to an extended version of ER diagrams. Shown in Figure 3-8 is a UML class diagram for Employment.

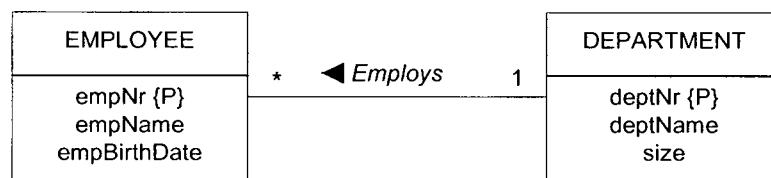


Figure 3-8: Unified Modeling Language Class Diagram

Classes are shown as named rectangles with the class name in the top compartment and attributes listed inside the rectangles. Entity instances in UML are identified by internal object identifiers, thus no conceptual identification schemes are required for its classes. In the above example, a user-defined constraint "P" has been added in braces to denote primary uniqueness. The uniqueness constraints on the Employee/Department association are captured by the multiplicity constraints \* (0 or more) and 1 (exactly one).

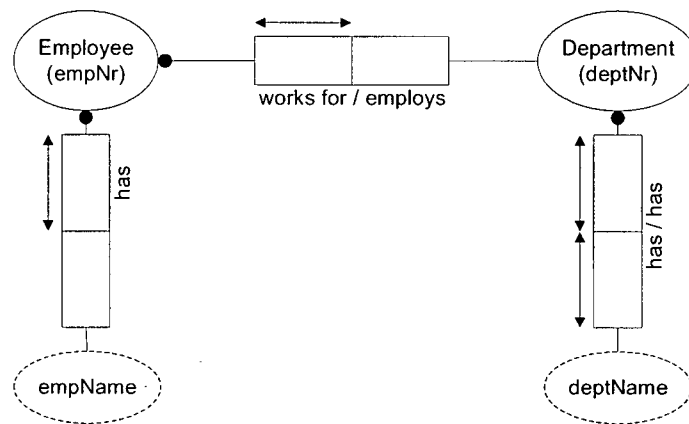
### 3.3.3. Fact-Oriented Modeling

At the heart of fact-oriented modeling is the verbalization of facts and rules, which facilitates the validation of business rules. Typically, a modeler develops a data model by gathering requirements from domain experts and communicating data structures at a conceptual level in terms that non-technical users can understand. To simplify things, the modeler usually breaks the information into manageable parts and

works with sample data populations. Fact-oriented modeling improves the analysis and communication required by verbalizing relevant data as elementary facts that cannot be split into smaller facts without losing information.

Our treatment of fact-orientation focuses on Object Role Modeling (Halpin, 2001) since this is the only fact-oriented method with significant support in the industry. ORM began in the early 1970s as a semantic approach that represents the application world as a set of objects (entities or values) that play roles (parts in relationships). ORM has appeared in many forms including Natural-Language Information Analysis Method (NIAM) from which it derives many of its features. The version we discuss is based on an extended version of NIAM called Formal ORM (FORM) and is supported by several industry tools including Microsoft's VisioModeler, Visio Enterprise, and Visual Studio.Net for Enterprise Architects.

ORM is used to create a conceptual schema where the schema specifies the information structure of the application: the *types of facts* that are of interest; *constraints* on these facts; and the *derivation rules* required for deriving some facts from others. Figure 3-9 shows a simple ORM schema. The ovals in the diagram represent *object types*. These are connected by *predicates*, shown as sequences of boxes. Each box corresponds to a role in the relationship. If we include the object types with the predicate, we have a *fact type* - for example, Employee works for Department. In this model, each predicate has two roles. For example, the fact type Employee works for Department has one role played by Employee (works for) and one role played by Department (employs).



**Figure 3-9: Object Role Modeling Schema**

The schema in Figure 3-9 also includes business rules, otherwise known as constraints or derivation rules. For example, an arrow-tipped bar over a role is a uniqueness constraint, indicating that each object playing that role does so only once (for example, each Employee has at most one empName). A dot on a role connector indicates the role is mandatory (each Employee must have an empName).

ORM simplifies the design process by using natural language, diagrams and examples, and by examining information in terms of elementary facts. Facts and rules can be easily verbalized as sentences and all data structures can be easily populated with multiple instances. Unlike ER or UML, no use is made of attributes so there is no need to determine whether a feature is to be modeled an attribute or not. This results in more stable models and queries that are more immune to attribute changes (Halpin, 2001).

### **3.3.4. Conventional Data Modeling and Data Warehouses**

Having described conventional data modeling approaches, we now highlight the data structure differences of traditional OLTP systems and decision support systems in order to distinguish conventional data modeling from multidimensional data modeling.

Data warehouses and OLAP applications contrast operational databases that support daily operations and on-line transaction processing. As proposed by Wu and Buchmann (1997), the major differences between operational databases and data warehouses include users, functionality, contents, and requirements. These differences are summarized in Table 3-1.

Aspect	Operational Databases	Data Warehouses
User	System Designer, System Administrator, Data Entry Clerk	Decision Maker, Knowledge Worker, Executives
Function	Daily Operations, (On-Line) Transaction Processing	Decision Support, (On-Line) Analytical Processing
DB Design	Application Oriented	Subject Oriented
Data	Current, Up-to-date, Atomic, Relational, Isolated	Historical, Summarized, Multidimensional, Integrated
Usage	Repetitive, Routine	Ad Hoc
Access	Read/Write, Simple Transaction (usually 1-3 tables)	Mostly Read, Complex Query (usually more than 3 tables)
System Requirements	Transaction Throughput, Data Consistency, Data Accuracy	Query Throughput, Data Accuracy

**Table 3-1: Operational Database and Data Warehouse Differences**

Adapted from "Research Issues in Data Warehousing," by M. Wu and A. Buchmann, 1997, *Proceedings of the 7th German Conference on Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'97)*, p. 62.

OLTP systems impose different requirements than data warehousing and OLAP systems, and therefore, different data models and implementation methods are required for each type of system. Conventional approaches like ER, UML, and ORM are commonly used to represent an OLTP application at the conceptual level, however, they are not capable of sufficiently representing multidimensional semantics using existing constructs. While the ER model is well proven as a powerful modeling technique for transaction processing systems, its deficiencies in modeling data warehouses have been well documented (Golfarelli, Maio, & Rizzi, 1998a; Kimball, 1997; Raden, 1995). As Kimball et al. suggest (1998), conventional data modeling cannot be used as the basis for enterprise data warehouses as it does not model the business, rather it models the micro relationships among data elements.



The main reason for the deficiency in conventional approaches is their underlying normalization premise. This premise provides for an efficient means to store data but does not satisfy retrieval requirements for analytical and decision support applications. As argued by Kimball et al. (1998), conventional models should not be used as the basis for data warehouses because the atomic detail of normalized models often confuses users and cannot be easily navigated by analysis tools. End users cannot understand, remember, or navigate conventional models and, unlike OLAP applications, there are few graphical user interfaces that can make conventional models usable by end users. Additionally, analysis software cannot usefully query general conventional models since optimizers that attempt to do this often make the wrong choices and result in poor performance.

Conventional models are constituted to remove redundancy in the data model and optimize OLTP performance by facilitating retrieval of individual records having certain critical identifiers. While these approaches are popular with OLTP systems, databases created using conventional modeling techniques cannot be efficiently queried. These techniques defeat the basic attraction of data warehousing, which is intuitive and high-performance retrieval of data, and should not be used for this purpose. As discussed in Bulos (1996), conventional models like the ER provide no easy way of modeling multidimensional data.

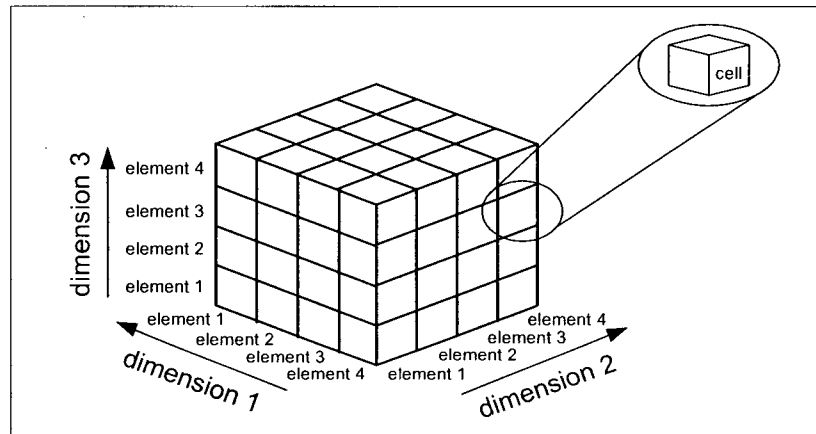
As a direct response to the inability of conventional models to service decision support processing, the multidimensional view of data arose as a popular alternative to conceptualize decision support data. Consequently, multidimensional modeling has emerged as the dominant technique for data warehouse design. Using the denormalized

data structures that result from multidimensional modeling, decision support applications run faster and employ a high level of data redundancy.

In addition to understanding the business rules within operational data, multidimensional modeling requires a good understanding of the analysis scenarios of an organization. To facilitate such an understanding, multidimensional modeling has introduced a variety of new modeling concepts, including facts and dimensions, classification hierarchies, derived measures, additivity, and categorizing dimensions. The following section provides an overview of these multidimensional concepts and discusses, through an example, a set of multidimensional modeling requirements needed to efficiently design data warehouses.

### **3.4. Multidimensional Data Modeling**

The multidimensional view of data is a popular conceptual view that influences front end tools, database design, and query engines (Chaudhuri & Dayal, 1997). As suggested by Boehnlein & Ulbriche-vom Ende (1999), the basic idea of the multidimensional view is the separation of quantitative and qualitative data. The quantitative measurable data, called *measures*, are analyzed from various viewpoints based on the qualitative content of the data, referred to as *dimensions*. These dimensions have been defined as the “combination of some qualitative aspects to a common structure” (Boehnlein & Ulbriche-vom Ende, 1999, p. 16). Together these lead to an n-dimensional structure, often visualized using the cube metaphor as shown in Figure 3-10.



**Figure 3-10: Representation of the Cube Metaphor**

A multidimensional cube corresponds to a subject of analysis commonly referred to in data warehousing terminology as a *fact* (e.g. a sales transaction). We do not consider the use of the term “fact” semantically correct in this context since all objects playing roles are essentially facts about the application domain. To avoid confusion, throughout the remainder of this paper we will use the more semantically correct term “event” to represent the subject of analysis instead of the more popular term “fact”. While this is a significant departure from the generally accepted use of the term, in light of our fact-oriented approach we feel it is necessary to reflect semantics accurately.

In the multidimensional model, every dimension has a set of elements called attributes. Shown on the axes of the cube in Figure 3-10, dimension attributes represent different ways of analyzing the data (e.g. store and time of purchase). The intersection of a dimension attribute for every dimension in the cube forms a cell containing a quantitative measure that describes the event (e.g. sales quantity). In most applications different measures describing an event are common, which means that a cell of the cube contains more than one numeric value. Multidimensional models are usually organized in terms of dimensions of the data, which are the terms of reference by which measures are

retrieved based on specific values of the dimensions. Dimension attributes can be arranged hierarchically and measures can be summarized along these hierarchies based on mathematical rules ranging from simple summarizations to complex averaging.

### **3.4.1. Multidimensional Concepts Through an Example**

In the following sections we discuss a set of multidimensional modeling requirements to understand the multidimensional concepts our fact-oriented model must accommodate to efficiently design data warehouses. Ranging from fundamental to advanced, these concepts represent the analytical processing requirements of end users.

To best understand multidimensional modeling and the structural properties of multidimensional data we draw from a well-known Grocery example (Kimball, 1997). We modify this example and reference it throughout the remainder of the paper to help describe our approach. A summary of our example is presented below.

*A large grocery chain with 100 stores is spread over a five-state area. Each of the stores has a full complement of departments, including grocery, frozen foods, dairy, meat, produce, bakery, floral and health/beauty aids. Each store has roughly 60,000 individual products on the shelves, each with bar codes referred to as stock keeping units (SKUs). As customers purchase products at the cash register, sales data is gathered by scanning bar codes into a point of sales (POS) system.*

*Management is interested in understanding customer purchases as captured by the POS system and they have decided to analyze the POS Retail Sales process. They hope to understand which products are selling to which customers at which stores during which times.*

### 3.4.2. Business Processes

As described by Kimball and Ross (2002), a *business process* is a major operational activity supported by a source system (e.g. invoicing) from which data can be collected for the analytic purposes of the data warehouse. Using that definition, the business process we wish to model for our Grocery example is *POS Retail Sales* since we are interested in customer purchases as captured by our POS transaction system.

### 3.4.3. Events

We define an *event* as an item of interest (e.g. a sales transaction) in a decision making process. Central to data warehouses, events are described through attributes called *measures*. Either atomic or derived, measures are numerical, continuously valued attributes that describe the event from different points of view. In our example, *Sales Transaction* is the event signifying the sale of an item and it is represented by the most granular data accessible to us - an individual line item on a POS transaction.

Sales measures include *price* (i.e. amount charged), *cost* (i.e. amount an item cost), and *profit* (i.e. the difference between the sale price and cost of the item).

#### 3.4.3.1. Derived Measures

While the majority of measures are atomic, we must also be able to model *derived measures*. A derived measure is one that is defined in terms of other measures, either atomic or derived. In our example, *profit* is derived as the difference between price and cost. When measures are arithmetically computed from others in this fashion we must also be able to include the appropriate mathematical calculations.

#### 3.4.3.2. Additivity

Measures are usually summarized in various ways in order to analyze information. Commonly known as *additivity*, this summarization refers to the ability to aggregate measure values along all hierarchies defined on a dimension. In our example, the *quantity* is additive as it can be summarized as the number of units sold for a product.

For other *non-additive* and *semi-additive* measures, aggregation is inherently impossible or limited in the context of one or more dimensions for conceptual reasons. The aggregation of some measures might not be semantically meaningful for these measures along all dimensions. An *inventory level* measure for example is non-additive on all the dimensions, since adding up levels does not make sense. However, we can still aggregate this non-additive measure using operators such as average, maximum, minimum.

Using our Grocery example, the *number of customers* is calculated for a given product, day and store by counting the number of tickets for a certain product printed on a certain day in a certain store. Since the same ticket may include other products, adding or averaging the number of customers for two or more products would be inaccurate. Thus, the number of customers is semi-additive as it cannot be consistently aggregated on the Product dimension, but is additive on the Time and the Store dimensions.

Our multidimensional data model should correctly summarize and produce results that are meaningful to the user when aggregating data. Specifically, it should avoid double-counting data and provide a foundation for specifying which summarizations are meaningful for different kinds of data. This concept of applying additivity to measures along dimensions is crucial to multidimensional data modeling.

### 3.4.4. Dimensions

Our measures are based on a set of *dimensions* that present the context for analyzing events. Dimensions contain discrete *dimension attributes* that characterize the dimensions and determine the minimum granularity chosen to represent events. In our example the dimensions for the Sales Transaction event are *Product*, *Customer*, *Store* and *Time*. Focusing on the Customer dimension, a customer may have dimension attributes including *customer id*, *name*, and *income level*.

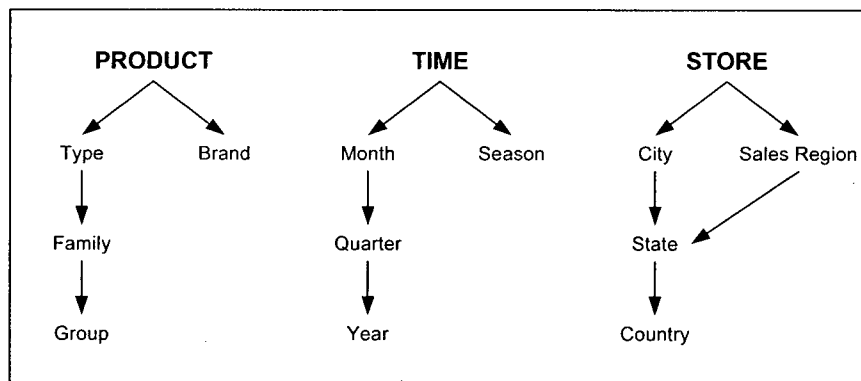
#### 3.4.4.1. Classification Hierarchies

Our multidimensional model must enable *classification hierarchies*. Hierarchies are made up of discrete dimension attributes and determine how measures may be aggregated and selected for the decision-making process. The dimension in which a hierarchy is rooted defines its finest aggregation granularity; other dimension attributes define progressively coarser granularities. Defining the classification hierarchies of certain dimension attributes is crucial because these classification hierarchies provide the basis for the subsequent data analysis.

A *hierarchy level* contains a distinct set of members and should be captured explicitly by our model so users can determine the relation between different levels in the hierarchy. Different levels correspond to different data granularities and ways of classification. Level A rolls up to level B if a classification of the elements of A according to the elements of B is semantically meaningful to the application. In our example, the Time dimension can be decomposed into *year*, *quarter*, *month* and *day* levels, showing that the day level rolls up to the month level, which rolls up to quarters,

which rolls up to years. In this way, product sales can be summarized on numerous levels for each year.

Since a level can roll up to any number of levels, our model must allow a single dimension to contain *multiple hierarchies*. This case occurs if different criteria of classification are possible for dimension members. Our model must not limit the number of hierarchies in a single dimension and it must support hierarchies with any different number of levels. Hierarchies may also share one or more common levels or attributes but otherwise have no correlation. Figure 3-11 shows the different classification hierarchies defined for the Product, Time, and Store dimensions. On the Product dimension, we have defined a multiple classification hierarchy so we can aggregate data values along two independent hierarchies: *product-type-family-group* and *product-brand*. In our Time example, we define multiple the hierarchies *time-month-quarter-year* and *time-season*.



**Figure 3-11: Examples of Classification Hierarchies**

As there can be more than one path along which to aggregate data in a single dimension, our model must also support *alternative path* hierarchies. This type of hierarchy occurs when several roll-up paths exist between two levels of a dimension. For



our Store dimension in Figure 3-11, we have defined an alternative path classification hierarchy with two different paths that converge into the same state hierarchy level: *store-city-state-country* and *store-sales region-state*.

#### 3.4.4.2. Strictness & Completeness

While we have presented several basic characteristics of dimension hierarchies, the concepts of *strictness* and *completeness* are also important for conceptual purposes. Strictness is used to mean that an object at a hierarchy's lower level belongs to only one higher level object. In our Store dimension example, Store and City have a strict relationship because a Store can exist in only one City. Similar strict relationships exist between City and State, with a City existing in only one State, and between State and Country, with a State existing in only one Country.

*Non-strictness* means an object belongs to more than one higher-level object. In a non-strict dimension hierarchy, many-to-many relationships may exist between the different levels in the dimension. The *Sales Region* and *State* objects, for example, form a non-strict relationship because a *Sales Region* can be comprised of more than one *State*.

Completeness means all hierarchy levels belong to one higher-class level and that level consists of those members only. In our example, we have a complete classification hierarchy between the State and Country levels since only the recorded States can form a Country. By this we mean, all the recorded States form the Country, and all the States that form the Country have been recorded. As another example, we may also define completeness for the Quarter and Year hierarchy levels because all the recorded Quarters form a Year, and all the Quarters that form the Year have been recorded.

#### 3.4.4.3. Categorization of Dimensions

While classification hierarchies provide a degree of distinction between hierarchy levels, as the number of dimensions increases certain attributes are valid for all hierarchy levels within a dimension while others are only valid for a subset of levels. Distinguishing this subset of attributes is referred to as the *categorization of dimensions* and must be supported in our model to situations like heterogeneous products.

In our Grocery example, we need to track a number of different products together with a common set of attributes and measures, but at the same time need to describe additional details about individual products. For example, our Product dimension attributes *alcohol percentage* and *volume* are valid for *Drink* products but are not applicable for *Food* products. Our multidimensional data model should consider these attributes and reflect the categorization semantics of the Product dimension.

#### 3.4.4.4. Many-to-Many Relationships Between Events and Dimensions

We usually consider events as many-to-many relationships between all dimensions and as many-to-one relationships between the event and every particular dimension. In our example, a sales transaction is related to a single product sold in one store to one customer at one time (e.g. a ticket line item). In some cases, however, events can represent *many-to-many relationships* between particular dimensions.

As seen in section 3.4.3.2, the reason for the non-additivity of number of customers on the Product dimension is that the relationship between purchase tickets and products is many-to-many instead of many-to-one. The sales and tickets form a many-to-many relationship to the Product dimension because one ticket can consist of more than one product, although every ticket is still purchased in only one store by one customer at

one time. Our modeling approach should semantically support the relationship between an event and a dimension as not always being the classical many-to-one mapping via constructs that are not traditional events or dimensions.

#### 3.4.4.5. Degenerate Dimensions

In our Grocery example, the grain of our sales event is a Sales Transaction as represented by a line item on a sales ticket. While the ticket is an identifying attribute for the Sales Transaction event, it has no other attributes that would make it an actual dimension and hence it is not treated as one. As Kimball and Ross (2002) indicate, this situation often arises when the grain of an event is represented on an actual working document such as an order or invoice. In these cases, order and invoice numbers often become *degenerate dimensions*.

### **3.5. Related Multidimensional Data Modeling Work**

Numerous multidimensional surveys exist that define specific requirements for multidimensional modeling and proceed to evaluate a series of models. Blaschka, Sapia, Hofling, and Dinter (1998) list requirements for a formal OLAP application model to analyze various models containing some kind of formalism. In a similar fashion, Pedersen and Jensen (1999) present requirements found in clinical data warehousing for multidimensional data models and evaluate several data models against them. While different models are compared, the models are relevant to different modeling phases and thus it is inappropriate to directly compare them.

In the remainder of this section we separate and discuss multidimensional modeling works based on the three information levels discussed earlier in the chapter. Our review is primarily concerned with translating an understanding of analytical

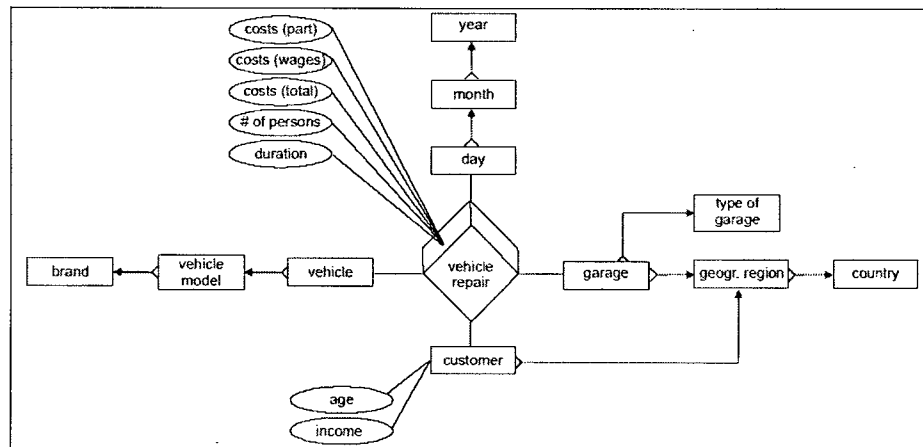
business requirements into a conceptual data warehouse design. As such, we focus the majority of our review on works at the conceptual level while briefly discussing other logical and physical design works for additional background. In addition to the three traditional information levels presented, we briefly mention a fourth group of models to complete our review. Categorized as Formal by Abello et al. (2001), these models are not specific to a particular database design phase; instead they provide a theoretical framework and multidimensional algebra or calculus.

### **3.5.1. Conceptual Level**

The following sections provide a general overview of existing models that attempt to capture multidimensional semantics at the conceptual level. While numerous models exist, we mention only the most relevant conceptual models found in the literature.

#### **3.5.1.1. Multidimensional Entity Relationship Model (M/ER)**

Sapia, Blaschka, Hofling, and Dinter (1998) propose the Multidimensional Entity Relationship Model (M/ER) as a specialization of the ER model. Illustrated in Figure 3-12, the M/ER model includes a special dimension level entity set (e.g. vehicle) and two special relationship sets connecting dimension levels - a fact relationship set (e.g. vehicle repair) and a rolls-up-to relationship set (e.g. vehicle-vehicle model). A rolls-up-to relationship set relates two dimension levels where the second one represents a higher level of abstraction. Multiple hierarchies, alternative paths, and shared hierarchy levels for different dimensions are supported and a fact relates different dimension level entities. While many constructs are supported, the M/ER model does not depict derived measures and their derivation rules, many-to-many relationships, strictness and completeness.

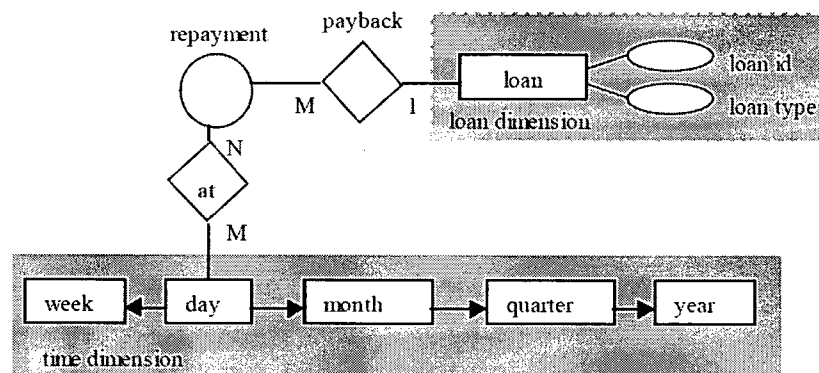


**Figure 3-12: Sample M/ER Model for Vehicle Repair**

Adapted from "Extending the ER Model for the Multidimensional Paradigm," by Sapia et al., 1998, *Proceedings of the 1st International Workshop on Data Warehouse and Data Mining (DWDM'98)*, p. 112.

### 3.5.1.2. Star Entity Relationship Model (starER)

Tryfona, Busborg, and Christiansen (1999) propose the Star Entity Relationship Model, basing their work on the ER model and the star schema. Shown in Figure 3-13, the starER includes fact sets, entity sets, relationship sets, and attributes. A fact set (e.g. repayment) represents a set of real-world facts sharing the same properties. An entity set (e.g. loan) represents real-world objects with similar properties, and a relationship set (e.g. at) represents a set of associations among entity sets and fact sets. Attributes (e.g. loan id) represent static properties of entity sets, relationship sets, and facts sets.



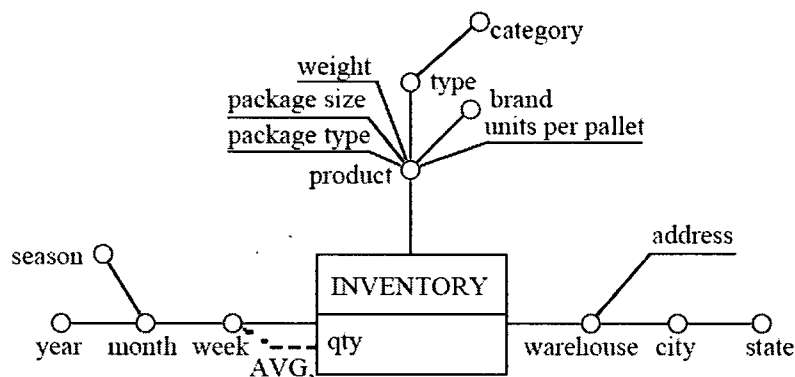
**Figure 3-13: Sample starER Model for Mortgage Repayment**

Adapted from "starER: A Conceptual Model for Data Warehouse Design," by Tryfona et al., 1999, *Proceedings of the 2nd International Workshop on Data Warehousing and OLAP (DOLAP'99)*, p. 6.

The starER's deficiency concerns the particular models used as its basis. Although the ER model is the most popular for transaction processing, its inadequacy in analytical processing is well documented as discussed earlier in this chapter. Although the star schema provides abstraction understandable to the user, it is still logical, not conceptual, and critical multidimensional semantics (e.g. derived measures) are lost.

### 3.5.1.3. Dimensional Fact Model (DFM)

Golfarelli et al. (1998b) propose a graphical conceptual model for data warehouses called the Dimensional Fact Model (DFM). Shown in Figure 3-14, the main components of the DFM are facts, measures, dimensions and hierarchies – together forming a fact scheme with the fact as root. A fact (e.g. inventory) is central to the DFM and its attributes are called measures (e.g. qty). Dimensions (e.g. product) are discrete attributes that determine the minimum level of granularity chosen to represent the fact. A hierarchy is a set of dimension attributes (e.g. type-category) linked by 1:1 or n:1 relationships and it may also contain additional descriptive information not used for aggregation. Additivity is expressed by relationships between a measure and a dimension, as tagged by the allowed aggregation functions.



**Figure 3-14: Sample Dimensional Fact Model for Inventory Management**

Adapted from "The Dimensional Fact Model: A Conceptual Model for Data Warehouses," by Golfarelli et al., 1998, *International Journal of Cooperative Information Systems*, 7(2-3), p. 226.

While the DFM accommodates the basic elements of multidimensional analysis and provides fundamental multidimensional constructs, is not well suited to express the complex properties of multidimensional data. Only many-to-one relationships between dimensions and facts are supported, objects not in the form of a dimension (i.e., not connected directly to a fact) cannot be modeled, and finally, there is no way to depict specialized relationships (e.g. specialization/generalization, membership).

The DFM also assumes a well-conceived relational model of source systems exists. As discussed by Boehnlein and Ulbriche-vom Ende (1999), a major disadvantage of this approach is having to first find a point of reference for the derivation of an ER diagram. This is especially true if the underlying models are very complex.

#### 3.5.1.4. GOLD Model

Trujillo, Palomar, and Gomez (2000) describe the GOLD model as an object-oriented conceptual model based on a subset of UML. A fact (e.g. sales) is represented as a basic class and is described through a set of fact attributes representing its measures (e.g. qty, price). Through shared aggregation, a fact is related to a set of dimensions (e.g. customer, time) that show the granularity adopted for representing facts.

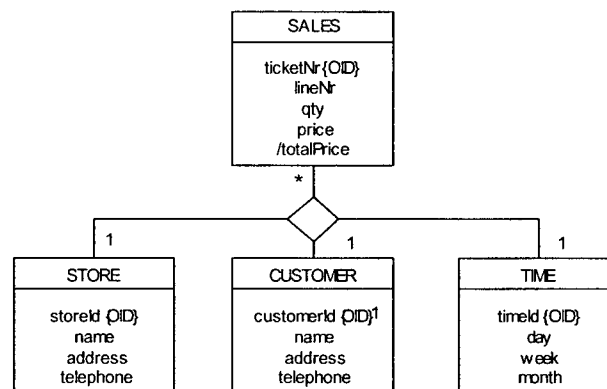


Figure 3-15: Sample GOLD Model for Retail Sales

The GOLD model is one of the most complete graphical conceptual modeling techniques found in the literature. It takes into account many of the fundamental elements of multidimensional analysis, including multiple classification hierarchies, strictness and completeness, additivity, and derived measures. However, the model is based on the graphic representation of UML, which is not well suited to conceptual data modeling (Halpin, 2001). Hay (1999) makes a similar argument in suggesting UML is not suitable for analyzing business requirements in cooperation with users.

### 3.5.2. Logical Level

Golfarelli et al. (1998a) discuss how the multidimensional model may be mapped to the logical level differently depending on the underlying data store. If the store supports multidimensionality, the model may be represented in a multidimensional database in an n-dimension array. Alternatively, in relational databases the model is usually mapped through a star schema as shown in Figure 3-16.

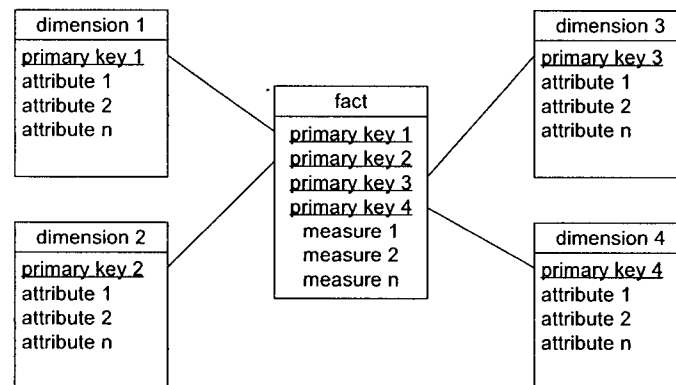


Figure 3-16: Logical Star Schema for Relational Databases

Undoubtedly the most well known logical model, the star schema is a result of the dimensional modeling technique made popular by Kimball (1996). Illustrated in, the star schema is a logical representation of multidimensional data structures in relational



database system. The two basic constructs that provide for multidimensionality in the star schema are fact and dimension tables. The primary key of the fact table is composed of a set of foreign keys to each one of the primary keys of the dimension tables.

As defined by Kimball (1996), dimensional modeling is a logical design technique prominent in data warehousing that is different from, and contrasts with, ER modeling. The technique seeks to present data in a standard, intuitive framework that allows for high performance access. As argued by Kimball, significant advantages of this model are it is highly recognizable to end users, it is a predictable framework that withstands unexpected changes in user behavior, it is gracefully extensible, and it handles common modeling situations. Unfortunately, although widely used, the technique is still logical in nature and multidimensional semantics (e.g. hierarchies) are not supported.

### **3.5.3. Physical Level**

Much of the concentration at the physical level is on specific storage techniques for particular DBMS implementations. Dyreson (1996) explains how a sparse cube could be implemented in a MOLAP database by means of cubettes but few constructs are provided. Theodoratos and Sellis (1999) investigate the problem of designing a data warehouse based on view materialization modeled as a search space problem. Other Physical works deal with indexing (Chan & Ioannidis, 1998), query evaluation (Cabibbo & Torlone, 1999), and query languages (Gingras & Lakshmanan, 1998).

### **3.5.4. Formal Level**

The focus of formal models is not on conceptualizing user ideas so they do not pay much attention to capturing specific user concepts. Instead, they are mainly devoted to the definition of a multidimensional algebra or calculus and do not offer as many

constructs as other models. Since our focus is on modeling constructs, formal models are not considered as conceptual, however, if we were to take into account the expressiveness of the algebras, they could certainly be as semantically expressive as conceptual models. Agrawal, Gupta, and Sarawagi (1997) presented one of the first formal multidimensional models. With a focus on presenting an algebra this model does not offer many conceptual elements to model a multidimensional scheme. In addition to Agrawal, another notable formal model is the Extended Multidimensional Data Model (EMDM) as proposed by Pedersen and Jensen (1999). The EMDM includes a multidimensional formalism and procedures are described for implementing the model using relational databases. Although these models support many complex multidimensional properties, information is not graphically represented in a conceptual schema.

#### ***3.5.5. Shortcomings of Existing Models***

To the best of our knowledge we have reviewed all of the proposed conceptual multidimensional models and the most relevant logical, physical, and formal models. We have found that conceptual models represent more semantics than models at other levels and there seems to be a trend to semantically enrich multidimensional models to overcome the limitations of conventional data models. However, while recent models are providing more functionality, they are still not ideal for formulating, transforming, and evolving a conceptual multidimensional model.

Table 3-2 presents the results of our conceptual multidimensional modeling review. In short, a complete and natural conceptual design technique was not found that adequately conceptualizes and clearly communicates multidimensional designs to business and technical users alike. While several models (e.g. GOLD, starER) were able

to represent fundamental event and dimension properties, all models lacked several desirable features. Specifically, we found existing approaches far removed from natural language and difficult to populate with sample, making it challenging for users and domain experts to conceptualize and validate designs. In addition, there is a general lack of design guidelines to ensure modeling approaches are properly and easily applied.

Multidimensional Modeling Criteria	Conceptual Multidimensional Models			
	M/ER	starER	DF	GOLD
<b>Events:</b>				
Atomic Measures	Yes	Yes	Yes	Yes
Derived Measures	No	No	No	Yes
Additivity	No	Yes	Yes	Yes
<b>Dimensions:</b>				
Classification Hierarchies	Yes	Yes	Yes	Yes
Strictness	No	Yes	No	Yes
Completeness	No	Yes	No	Yes
Categorization of Dimensions	Yes	Yes	No	Yes
Degenerate Dimensions	Yes	Yes	Yes	Yes
Many-to-Many Relationships	No	Yes	No	Yes
<b>Business Processes:</b>				
Business Process Families	No	No	No	No
Business Processes	Yes	Yes	Yes	Yes
<b>Other:</b>				
Natural (Fact) Basis	No	No	No	No
Population & Validation Mechanisms	No	No	No	No
Design Guidelines	No	No	Yes	No
Implementation Using Existing Modeling Tools	No	No	No	Yes
Generation into an OLAP Tool	Yes	No	No	Yes

**Table 3-2: Comparison of Existing Conceptual Multidimensional Models**

Despite the growth of data warehousing, a standard does not exist to indicate what should be represented in a multidimensional scheme. While it is widely recognized that data warehouses are based on the logical star schema, there is no standard conceptual data model commonly accepted for data warehousing and OLAP applications. Most of the models reviewed use their own terminology and define a specific set of design elements. Consequently, user analysis via a common framework is difficult and there is no consistent basis for solving conceptual multidimensional modeling problems with an intuitive and complete conceptual model.

### **3.6. Summary**

This chapter presented an overview of data modeling for OLTP systems, data warehouses and OLAP applications. We first examined basic data modeling concepts by looking at the conceptual, logical, and physical levels of data. We then presented an overview of several conventional data modeling approaches with a specific focus on ER Modeling, UML and ORM. Our review highlighted the differences between data warehouses and traditional OLTP applications and we concluded different conceptual modeling techniques are required for data warehouses due to the multidimensional nature of analytical data. To better understand multidimensional data requirements we presented various multidimensional concepts through an example.

Using the analysis requirements demonstrated with a sample Grocery chain, we covered events and dimensions, measures, additivity, derived measures, classification hierarchies, strictness, completeness and categorizing dimensions. The chapter concluded with a review of the current state of multidimensional modeling works with a focus on those attempting to express semantics at the conceptual level – the M/ER, starER, DFM and GOLD models. The fundamental deficiencies and shortcomings of these approaches in formulating, transforming and evolving a conceptual model provides motivation for our model presented in the next chapter. Inspired by ORM, our proposed approach considers an information system's structural properties at the conceptual level more naturally than existing multidimensional models or conventional modeling approaches.

## **4. FACT-ORIENTED MULTIDIMENSIONAL MODELING**

### **4.1. Introduction**

This chapter introduces Multidimensional Object Role Modeling (MORM), our fact-oriented approach to conceptually modeling multidimensional data. We will first present several key design considerations in specializing ORM and discuss the advantages and disadvantages of using ORM in our approach. We will also demonstrate how MORM easily represents the main structural properties of multidimensional data at the conceptual level. Our approach is presented in sections, each one outlining how our model addresses a multidimensional modeling concept as presented in the previous chapter. Our approach is a specialization of ORM in which we introduce several multidimensional constructs and provide semantics, syntax, and rules for each. We will also present design guidelines for our model in order to provide data modelers with a systematic approach to building a conceptual multidimensional model using our approach. The chapter concludes with an evaluation of our model and a discussion of its benefits with respect to multidimensional and conceptual criteria.

### **4.2. Key Design Considerations**

In order to allow the natural representation of semantics inherent in multidimensional data, we specialize ORM. We do not propose a set of new concepts and terminology, rather we attempt to pull multidimensional concepts together under the ORM framework to try and understand their semantics while keeping core ORM constructs the same. In doing so, our approach is driven by the following key design considerations:

1. Specialization of ORM: All newly introduced elements should be special cases of native ORM constructs. Thus, we maintain the flexibility and expressiveness of ORM.
2. Minimal extension of ORM: The number of additional elements needed should be as small as possible to ensure we can easily transfer scientific results from ORM to our model. Minimal extension also ensures an experienced modelers can easily learn and use our specialized model.
3. Representation multidimensional semantics: Our specialization should be powerful enough to express advanced multidimensional semantics, namely events and dimensions, additivity, derived measures, classification hierarchies, strictness, completeness and the categorization of dimensions.

### **4.3. Why Use Object Role Modeling?**

Prior to presenting our conceptual multidimensional modeling approach, we discuss several primary reasons for our use of ORM. Simply put, we contend that building a good data model requires capturing and expressing as much information as possible at the conceptual level and we believe ORM is the best way to do this. Building such a model requires an ability to first design a conceptual schema that accurately and completely defines business rules in a way business users understand. To do this we must effectively communicate with these users since we rely on them to define the rules that dictate and validate the data. The following sections further elaborate on our use of ORM, providing several arguments for and against its use as a modeling method.

#### **4.3.1. Advantages of Using ORM**

Designing a database requires a complete understanding of the subject area of interest and ORM allows us to specify this in a clear and unambiguous way. ORM uses natural language (e.g. English) and easily understood diagrams that are populated with sample data to accomplish this goal. Since ORM is based on natural language, it can be completely expressed in either graphical or textual format. This natural language is much

easier for users to understand, express, and verify than technical terminology and allows for communications with business experts in their own language.

Another significant advantage of ORM is that it makes no initial assumptions about an object's importance until performing conceptual to logical schema mapping. The foundation of ORM is the elementary fact through which the universe of discourse is expressed in terms of objects playing roles. Using simple, easy to understand facts like "Person works for Department" requires no distinction as to whether an object is an attribute or an entity and delays any commitment on the relative importance of each. Delaying the decision to model an element as an attribute or an entity allows us to be concerned only with the data and business rules and alleviates costly data integrity and schema change problems in the future.

The fact-based approach of ORM is a simple, accurate approach that makes it easy to apply a population check with real data that makes it easier to get one individual fact correct than many facts simultaneously. It is also easy to determine constraints while looking at sample data sets through ORM modeling. Semantic domains (e.g. units or ranges such as "name", "SSN", etc.) are automatically included in these data sets, meaning there is less chance for error in the final model.

#### 4.3.1.1. Conceptual Modeling Evaluation Criteria

Our reasons for using ORM are evidenced in ORM's evaluation results against a well-defined set of criteria for conceptual models - expressibility, clarity, semantic stability, semantic relevance, validation mechanisms, abstraction mechanisms and formal foundation (Halpin & Bloesch, 1999). Halpin & Bloesch suggest these criteria are

desirable characteristics for any language to be used for conceptual modeling. In support of our use of ORM, we summarize the results of their evaluation below:

- Expressibility of a language is a measure of what it can be used to say about a domain. For conceptual data modeling, ORM's rich constraint notation makes it expressive both graphically and through the use of textual languages. It has many constructs inherent to the language, and is therefore more expressive of the actual universe of discourse. Its role-based notation makes it easy to specify a wide variety of constraints, and its object types reveal the semantic domains that bind a schema together.
- Clarity of a language is a measure of how easy it is to understand and use. With respect to clarity, ORM structures may be directly verbalized as sentences and its notations and textual expressions are easily learned and remembered.
- Semantic stability is a measure of how well models retain their original intent in the face of changes to the domain. Attribute-free, ORM is more stable for modeling and not impacted by changes that would otherwise cause attributes to be remodeled as relationships or vice versa.
- Semantic relevance means only relevant conceptual details need be modeled. Using purely conceptual constructs ORM avoids modeling logical or physical aspects such as implementation details.
- Validation mechanisms are ways in which domain experts can check whether the model matches the application. ORM uses "data use cases" to initiate data modeling through the verbalization and population of facts and rules. Using simple sentences, this approach facilitates communication between data modelers and users so the domain is understood and the application model is validated.
- Abstraction mechanisms allow unwanted details to be removed from immediate consideration. ORM models may be modularized into various scopes or views based on perspective (e.g. a page of a data model). Other mechanisms like attribute abstraction can be used to hide or show only a portion of the model.
- Formal foundation of a language is needed to ensure it is executable and not ambiguous. ORM has a sound theoretical basis and a mature formal foundation that refines its semantics.

#### **4.3.2. Disadvantages of Using ORM**

While we have presented our case for the use of ORM, there are many arguments against its use. As described by Becker (2000), these arguments and their rebuttals are summarized as follows:



1. Standard industry CASE tools (e.g. ERwin, Data Architect) do not support ORM and ORM's CASE tools (e.g. Visio) are not enterprise level tools: While this is true, the importance should be on ensuring requirements are correctly, precisely, and accurately gathered and that the resulting design meets those requirements. It should not matter what tool is used to do this. If a project fails because the underlying data model is not correct, the tool, no matter how standard, is worthless. In cases where a model must absolutely go in a company standard deliverable tool, it can simply be entered once it has been developed in ORM and deemed to be correct.
2. ORM models are too verbose and take up too much space: ORM models are indeed verbose, mainly because they capture many constraints that other techniques are unable to express (e.g. attribute level constraints and set comparison constraints like subsets and exclusionary rules). As such, it is more important that the model completely specify the problem regardless of how much space it utilizes. In either case, compact versions of ORM models can be easily generated using ORM CASE tools.
3. Virtually perfect models can be created in ER and/or UML: This is true but using ORM's CSDP can make the process easier and lead to fewer mistakes. Like ORM modelers, ER and UML modelers basically think about objects and the roles they play in order to implement them correctly in terms of their methods (e.g. deciding what is an entity vs. what is an attribute). ORM just makes this process more formal. It is also important to note that ER models can be derived from ORM models relatively easily.
4. The world is going UML and we do not need yet another data-centric technique: ORM and UML are not mutually exclusive, rather they can be used together and the overall results are usually better. ORM is a natural fit into the UML process flow, particularly at the analysis stage where ORM can document the data and static constraints while UML can document processes and dynamic rules. Using these together can result in analysis deliverables that are better formed, more consistent, more accurate, and more concise.
5. Users won't understand yet another diagram type: In ORM's case users do not even need to see the notation if they do not want to. Since ORM is based on natural language, users can be shown sentences in English or whatever language they understand. While users often pick up on notations relatively quickly, they are often immediately comfortable with ORM's sentences and its narrative style of data use cases.

#### **4.4. Multidimensional Object Role Modeling (MORM)**

Having provided an overview of Object Role Modeling and multidimensional modeling concepts in the previous chapter, we now bring these two topics together with our fact-oriented multidimensional modeling approach. Based on our observation of the

limitations of existing conceptual techniques for multidimensional modeling, we propose MORM which introduces multidimensional constructs to ORM's grammar to support multidimensional semantics. We do not propose an entirely new set of constructs and terminology to represent these concepts, rather we utilize the ORM framework and specialize it as required to represent multidimensional modeling semantics.

Our design goal is to provide a simple yet powerful approach that represents multidimensional properties at the conceptual level. To achieve this we combine multidimensional constructs with the semantically rich constructs of the well-known ORM model as summarized in Appendix A. Our starting point is that ORM has been used productively for years and has tested powerful enough conceptually, that when new modeling techniques are needed to capture new demands, we should look to ORM.

To represent multidimensional properties at the conceptual level we introduce three specialized object types – the *Event Object Type*, *Dimension Object Type*, and *Hierarchy Object Type*. To distinguish these multidimensional constructs from native ORM and ensure they are emphasized in our models, a special graphical notation is defined for each as shown in Figure 4-1. These constructs are briefly defined in Table 4-1 and further described using examples in the sections that follow.

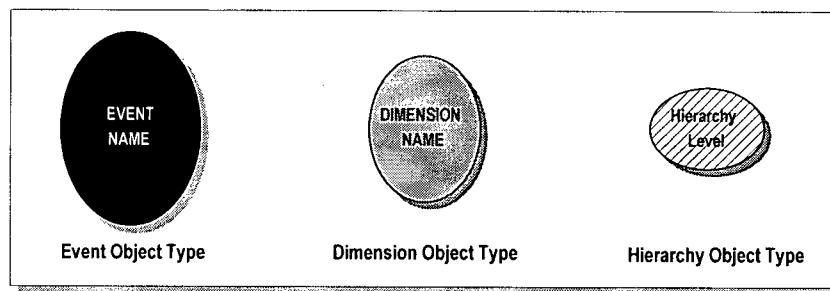


Figure 4-1: Graphical Notation For New MORM Constructs

<b>Construct</b>	<b>Description</b>
<i>Event Object Type</i>	Depicts an event (e.g. sales transaction) that is described with quantitative measurable data and analyzed in terms of dimensions. Depicted as a black filled inverted ellipse with the event name in white upper case lettering. It is larger than other constructs, signifying it is the focal point of analysis.
<i>Dimension Object Type</i>	Depicts a dimension (e.g. store) representing an analysis viewpoint based on the qualitative content of the data. Forms the root of a dimension tree, where each node is an object type and each edge is a functional (n:1 or 1:1) predicate. Depicted as a gray shaded inverted ellipse with black upper case naming. Its size and shading signifies its importance as an analysis viewpoint on an event and separates it from native ORM.
<i>Hierarchy Object Type</i>	Represents each classification hierarchy level (e.g. month) within a dimension (e.g. time). A role between two hierarchy objects specifies a relationship between two levels of a hierarchy (e.g. month is in quarter). Depicted as a gray slash-filled ellipse with mixed case naming, its fill signifies its importance in data analysis (e.g. aggregation) and separates it from native ORM.

**Table 4-1: MORM Constructs and Associated Descriptions**

Since our model is a specialization of ORM, regular ORM constructs as outlined in Appendix A are used in our MORM diagrams. The extended diagram that results from the combination of both techniques allows us to efficiently model both conventional concepts (e.g. value types, roles, etc.) and multidimensional properties (e.g. dimensions, hierarchies, etc.) at the conceptual level.

Our approach builds on ORM's conceptual schema design procedure (CSDP), a formal method for designing a conceptual schema from a universe of discourse (Halpin, 2001). Shown in Table 4-2, the CSDP focuses on data analysis and design through seven primary steps.

<b>Step</b>	<b>Description</b>
1	Transform familiar information examples into elementary facts, and apply quality checks
2	Draw the fact types, and apply a population check
3	Check for entity types that should be combined, and note any arithmetic derivations
4	Add uniqueness constraints, and check arity of fact types
5	Add mandatory role constraints, and check for logical derivations
6	Add value, set comparison and subtyping constraints
7	Add other constraints and perform final checks

**Table 4-2: Conceptual Schema Design Procedure (CSDP)**

Since we use ORM as the basis for our model, the fundamental CSDP steps still apply. However, we need to consider additional guidelines to incorporate multidimensional concepts as presented in the previous chapter. The following sections summarize how MORM represents these main structural aspects of multidimensional data.

#### 4.4.1. Business Processes

Consistent with Step 1 of ORM's CSDP, familiar examples of business process information from the application domain are initially gathered from reports, forms, the domain expert or other application documentation. Our approach then translates the information regarding the high-level relationship between business process events and dimensions into *elementary facts*.

Following our Grocery example presented in the previous chapter, we begin to gather information from our POS Retail Sales business process and verbalize it as facts f1 through f4 as shown in Table 4-3.

#	Fact
f1	The <i>Sales Transaction</i> with ticketNr 715 <b>occurred in</b> the <i>Store</i> named UBC Foodmart
f2	The <i>Sales Transaction</i> with ticketNr 715 <b>occurred at</b> the <i>Time</i> indicated 12:00
f3	The <i>Sales Transaction</i> with ticketNr 715 <b>included</b> the <i>Product</i> with product id 123456
f4	The <i>Sales Transaction</i> with ticketNr 715 <b>was completed by</b> the <i>Customer</i> with customer id 99

Table 4-3: Retail Point of Sale Facts

Facts 1 through 4 specify relationships between the Sales Transaction *event* and the *dimensions* Product, Store, Customer, and Time. As in ORM, each fact expresses a fundamental step in our MORM approach - "an object plays a role with another object". Facts assert that the objects participate in a *relationship* (play *roles*), where that relationship cannot be expressed as a conjunction of simpler facts. As with ORM fact assertions, object types begin with a capital letter and are displayed here in *italics*. The

relationship type, or logical *predicate*, is shown in bold between the noun phrases that identify the objects. For our purposes, only the normal predicate is included in the declarations. If the inverse was included it would be preceded by a slash "/". For example, in f4 the "/" would indicate that the Sales Transaction plays the role of *being completed by*, and the Customer plays the role of *completing*.

In the above case, the fact description indicates the *entity type* (e.g. Product), a *value* (e.g. 123456) and a *reference mode* (e.g. Product Id) indicating the manner in which the value refers to the entity. Removing the reference modes, the facts may also be stated as shown in Table 4-4.

#	Fact
f1	Sales Transaction 715 occurred in Store UBC Foodmart
f2	Sales Transaction 715 occurred at Time 12:00
f3	Sales Transaction 715 included Product 123456
f4	Sales Transaction 715 was completed by Customer 99

---

**Table 4-4: Facts with Reference Modes Omitted**

Stated even more briefly by removing the values, the above facts are instances of the *fact types* shown in Table 4-5.

#	Fact-Type
f1	Sales Transaction occurred in Store
f2	Sales Transaction occurred at Time
f3	Sales Transaction included Product
f4	Sales Transaction was completed by Customer

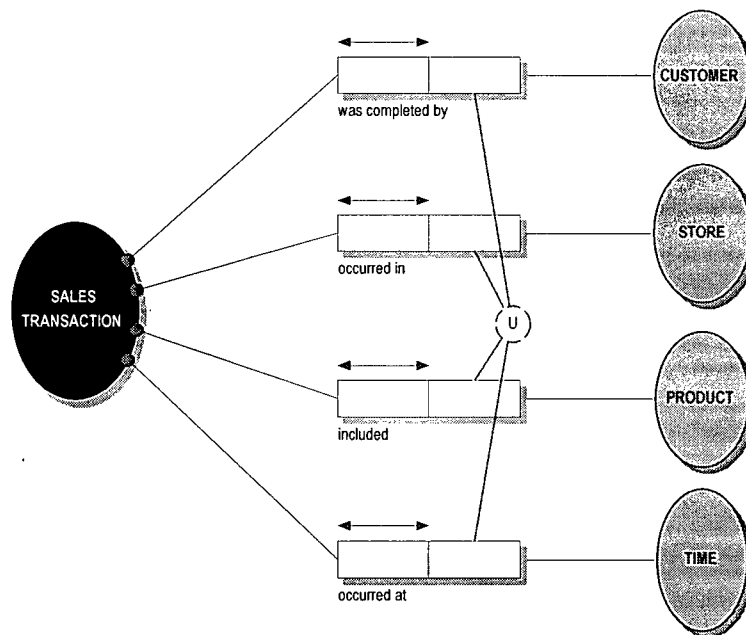
---

**Table 4-5: Fact Types with Values Omitted**

#### 4.4.1.1. Event & Dimension Constructs

Once business process information examples are translated into elementary facts a conceptual schema is drawn showing all the fact types. To support the multidimensionality inherent in information at the business process level, we introduce two ORM constructs - the *Event Object Type* and the *Dimension Object Type*. These constructs represent the events and dimensions we are interested in analyzing.

Figure 4-2 depicts the high-level ORM model for the POS Retail Sales business process. Our approach clearly divides business process data into events and dimensions, as is evident from the Sales Transaction event object and its relationship to the dimension objects Product, Store, Customer, and Time.



**Figure 4-2: Schema for POS Retail Sales Business Process**

The event object type is depicted as a black filled inverted ellipse. Its name is capitalized and it is larger than other constructs, signifying it is the focal point of analysis. Dimension object types are smaller inverted ellipses but are also shaded gray

and capitalized to distinguish them from other object types that will be added as the model progresses. Reference modes are intentionally omitted from the diagram at the business process level and are more appropriately included when event and dimension details are modeled. Lines connect dimension object types to the roles they play and predicates are shown as named sequences of two role boxes. Predicate names are read left-to-right, however, there is only one role name to read in this figure as inverse predicate names have been intentionally omitted.

*Mandatory roles* are explicitly shown by means of a mandatory role dot where the role connects with its object type. In our example, all roles for a Sales Transaction are mandatory, meaning all Sales must be associated with a certain Product, Store, Customer, and Time. Roles without a mandatory dot are optional, as seen by the inverse roles as read from the each of the dimensions. Although not included, an inverse role for our example could read Product *is included in* Sales Transaction. The optional inverse role indicates a dimension object can be part of zero, one, or more event object instances. In short, our example dimensions may exist without playing a role in a Sales Transaction.

ORM's *internal uniqueness* constraints are used on the binary fact types to capture cardinality by asserting entries in roles occur there at most once. For example, the internal uniqueness constraints (tipped arrows) on our binary fact types assert that each Sales Transaction occurred in at most one Store. This depicts a many to one relationship with the first role mandatory. The absence of a uniqueness constraint on dimension role indicates each Product can be part of many Sales Transactions. This absence may be expressed explicitly by the default verbalization "it is possible that the same Store sells more than one Sales Transaction.

The *external uniqueness* (circled “u”) constraint spanning roles of the different predicates to all dimensions specifies that in the natural join of the predicates, the combination of connected roles is unique. This stipulates that for each Sales Transaction, the combination of Product, Store, Customer and Time is unique. Stated another way, given any combination of the four dimensions there is at most one Sales Transaction.

#### 4.4.1.2. Families of Business Processes

Most organizations have an underlying value chain that represents the natural flow of key business processes. Operational source systems produce transactions or snapshots at each step of the value chain and generate interesting performance metrics along the way. Each key process produces distinct metrics with unique granularity, time intervals and dimensionality so each is typically modeled separately. As put forth by Kimball and Ross (2002), an Enterprise Data Warehouse (EDW) often integrates this set of related business processes based on common, shared dimensions. An integrated data warehouse combines measures from different processes to provide insight into performance across the value chain.

Our approach to multidimensional modeling ensures we accurately represent these “*families*” of business processes when modeling large, complex data warehouses. To illustrate this concept, we now widen our Retail example to include store inventory:

*Optimized inventory levels in our grocery stores can have a major impact on chain profitability. Making sure the right product is in the right store at the right time minimizes out-of-stock situations and reduces overall inventory carrying costs. To better understand the inventory-sales relationship, management would also like the ability to analyze daily quantity on hand inventory levels by product and store.*



In Figure 4-3 we illustrate the business process family concept using two business processes – Retail Sales and Inventory. The Sales Transaction and Inventory event objects represent metrics captured by these processes and share three common dimensions - Product, Time, and Store.

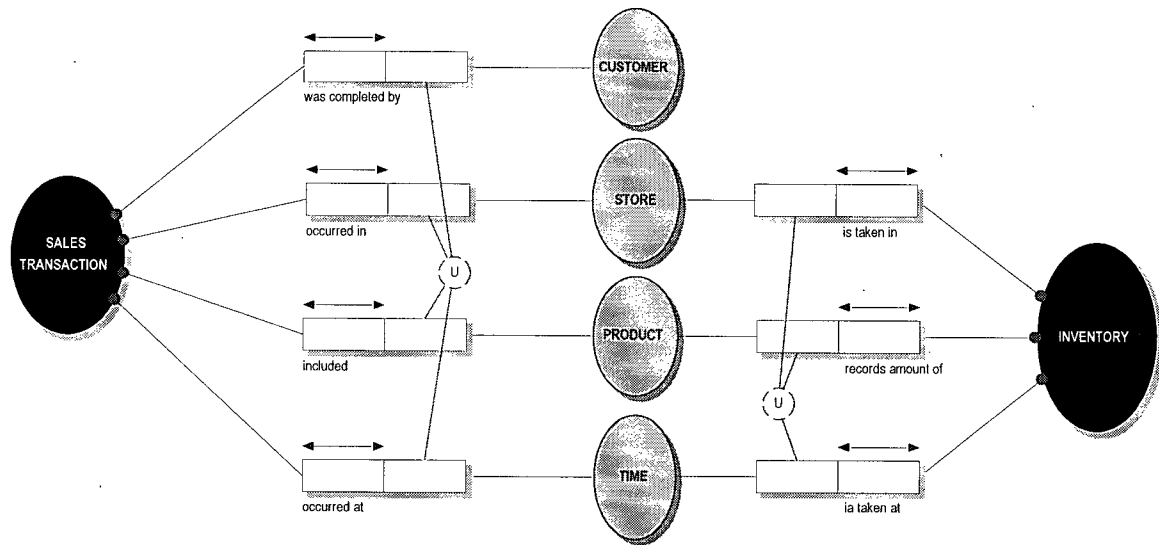


Figure 4-3: Schema for Retail Business Process Family

#### 4.4.2. Events

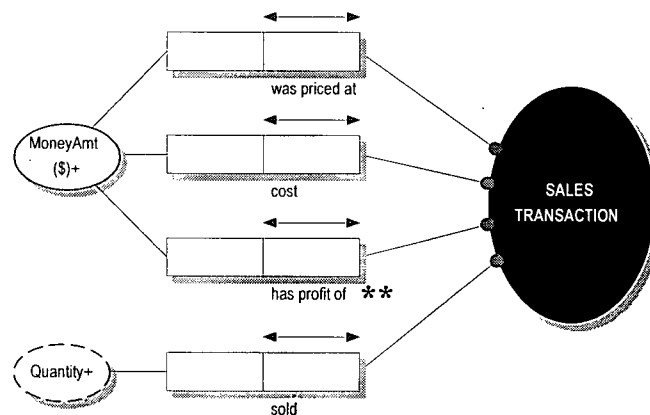
The following sections outline our approach to representing events in our ORM model. Following the concepts presented in chapter 3, major considerations for events include atomic measures, additivity, derived measures and many-to-many relationships between events and dimensions.

As the initial step in our approach, familiar examples of event information are gathered from the application domain, verbalized into natural language, and subsequently translated into elementary facts. Following our Retail example, information gathered for the Sales Transaction event is verbalized into the fact types included in Table 4-6.

#	Fact Type
f1	Sales Transaction cost MoneyAmount
f2	Sales Transaction was priced at MoneyAmount
f3	Sales Transaction had profit of MoneyAmount
f4	Sales Transaction sold Quantity

**Table 4-6: Sales Transaction Event Fact Types**

Once event information examples are translated into elementary facts we have a set of fact types that can now be refined for business rules (e.g. constraints and derivations) and added to the conceptual schema. A conceptual schema for our Sales Transaction event is shown in Figure 4-4. Our MORM approach illustrates relevant object types, predicates and reference schemes for the event.



**Figure 4-4: Schema for Sales Transaction Event**

#### 4.4.2.1. Atomic Measures

*Atomic measures* are those that are primitive, or not defined in terms of others. The atomic measures indicated in our Sales Transaction event are price, cost and quantity. Our approach uses the common object type MoneyAmt for the three monetary measures because we wish to make the domain explicit. This makes it clear that we can compare monetary values (e.g. price vs. cost). The broken ellipse for Quantity indicates this is a value type, in this case a number, and hence needs no reference scheme.

#### 4.4.2.2. Derived Measures

In our approach, *derived measures* are marked with an asterisk “\*” to indicate their derivability and distinguish them from atomic measures. When measures are arithmetically derived from others an appropriate mathematical calculation (referred to as a *derivation rule*) must be provided. This derivation rule references other fact types in the model. Similar to ORM, our approach uses a double asterisk “\*\*” to indicate that the *derivation rule* is to be added to the conceptual model and the associated measure is to be *stored* in the physical database. In this derived and stored case, a derived measure is stored as soon as its defining measures are entered in the database and it is updated whenever they are updated. Our approach includes both the derived fact type indicator (\*\*) and the rule for clarity. As shown in Figure 4-5, our example includes the derived measure *profit* and its derivation rule is written as text in the schema.

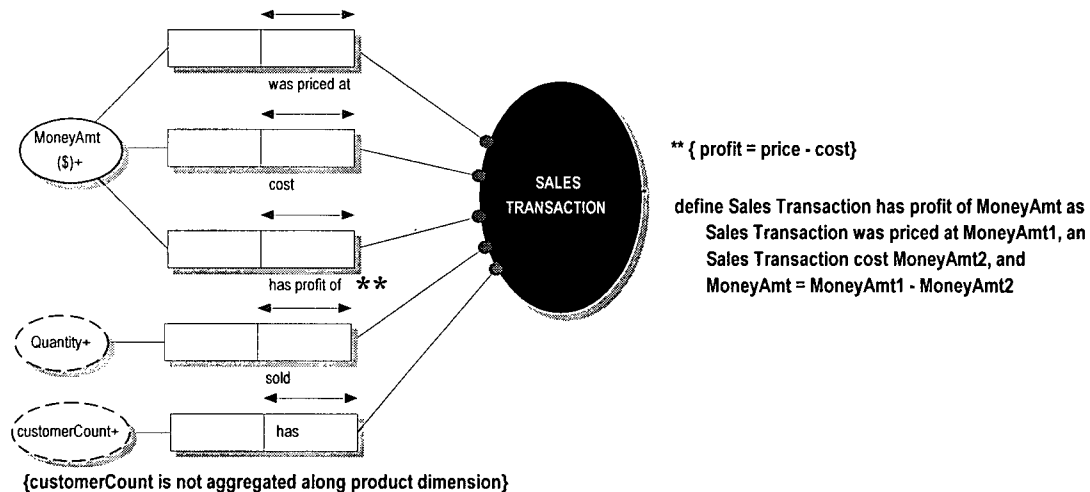


Figure 4-5: Derivation Rule for the Profit Measure

Different styles may be used in ORM but we use a *relational style* in which fact types are set out fully as relationship types. In this relational style predicates declare the rule. An informal version of the rule is written as a comment in braces, while a formal

version specifies a derivation rule in which the derived fact type is defined in terms of the others. In this definition, the derived fact type is said to be the definiendum, meaning what is required to be defined (Halpin, 2001).

Various textual languages have been defined to express constraints, derivation rules and queries in ORM schemas. We chose ConQuer, an ORM query language for embedding constraints in our conceptual model. ConQuer is essentially classical logic with set theory, and since an ORM fact table is a set of tuples, derivation rules can be expressed in ConQuer using set comprehension (Halpin, 2001).

#### 4.4.2.3. Additivity

Consistent with ORM, our approach uses a plus sign "+" to represent measures referenced by a number, thereby indicating they can participate in numeric operations. All measures with a "+" following their reference scheme are considered *additive*. For example, the "+" on Quantity in Figure 4-5 indicates that the values which refer to Quantity are actual numbers and hence may be added. *Non-additive* measures are not depicted with the "+" symbol.

For *semi-additive* measures we include an informal rule as a comment in braces. Shown in Figure 4-5, we include a rule for customerCount because it is additive on Time and Store dimensions but cannot be aggregated along Product since the same ticket may include other products.

#### 4.4.3. **Dimensions**

As with our previous business process and event domains, the initial step of our approach to modeling dimensions is to translate familiar information examples from the application domain into elementary facts. If these examples are verbalized for the

Product dimension we translate them into a base set of fact types that include the following examples in Table 4-7.

#	Fact Type
f1	Product <b>is identified by</b> Product Key
f2	Product <b>is known by</b> Product Name
f3	Product <b>is of</b> Product Type
f4	Product <b>belongs to</b> Product Group

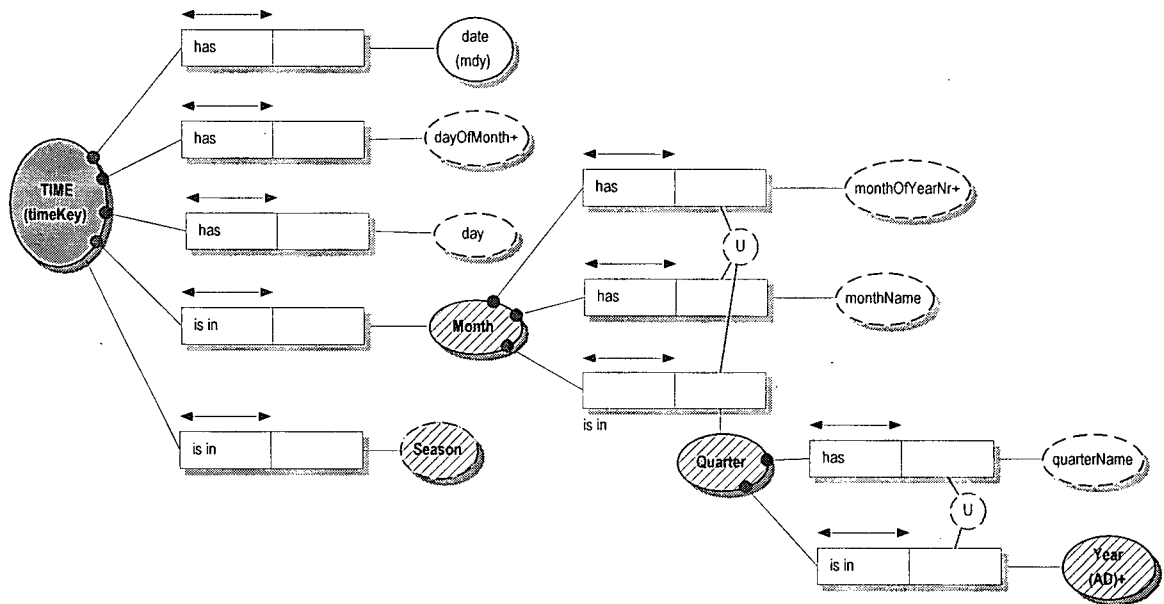
**Table 4-7: Product Dimension Fact Types**

As in our earlier examples, each sentence fact is expressed in plain language using a meaningful predicate and non-technical object names that can be mapped to technical database names later. Note that the addition of the "Product Key" provides a reference concept to the dimension to make each Product unique. Each dimension in our multidimensional model can be expressed using this same approach.

#### 4.4.3.1. Classification Hierarchies

Introduced earlier in this chapter, the Dimension object type forms the root of a dimension tree, where each node is an object type and each edge is a functional (n:1 or 1:1) predicate. To support semantics inherent in the dimension tree, we introduce a third ORM construct - the *Hierarchy Object Type*. Hierarchy object types represent each *classification hierarchy level* within a dimension and are depicted as named ellipses with lightly shaded slash-fill notation.

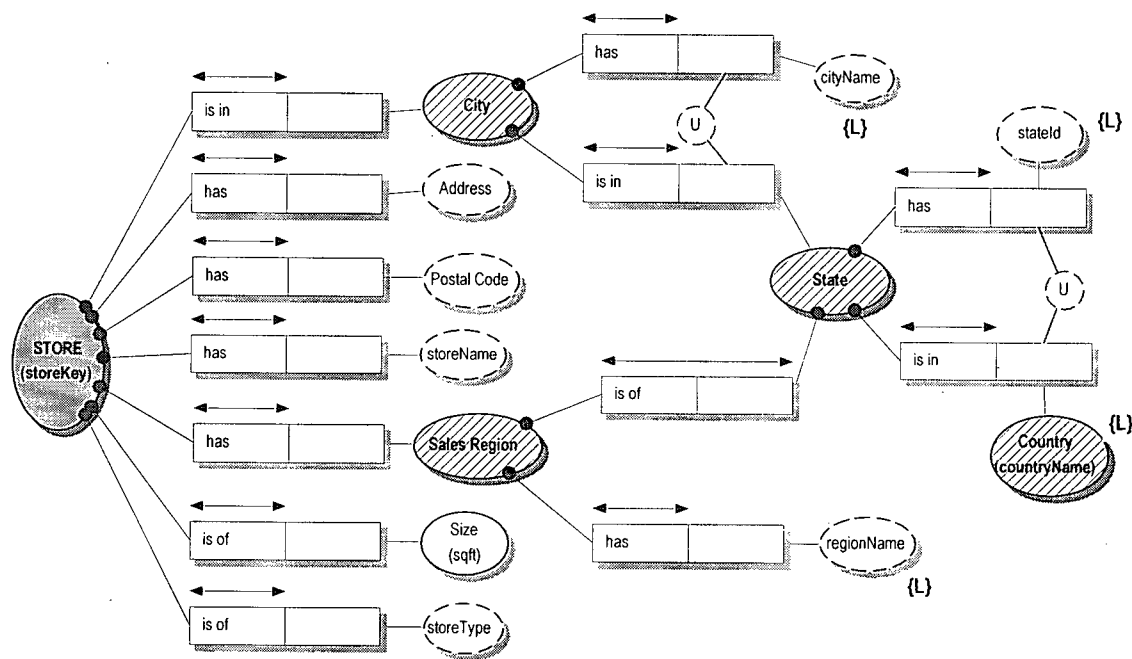
Figure 4-6 shows the classification hierarchies defined for the Time dimension. This schema illustrates how multiple classification hierarchies are possible using our Hierarchy object types, allowing us to aggregate event measures along two different hierarchy paths: *time-month-quarter-year* and *time-season*.



**Figure 4-6: Multiple Classification Hierarchies in ORM**

In our approach, a predicate between two hierarchy objects specifies a relationship between two levels of a classification hierarchy (e.g. Month is in Quarter). Other entity types and value types may play roles with hierarchy object types to provide additional information (e.g. Month has monthOfYearNr) but may not be used for aggregation purposes as a classification level hierarchy.

Our approach also uses hierarchy object types to model alternative path hierarchies with two different paths that converge into the same hierarchy level. In Figure 4-7 we depict the following alternative path hierarchies for the Store dimension: *store-city-state-country* and *store-sales region-state*.



**Figure 4-7: Alternative Path Hierarchies in ORM**

In our example, from the Store dimension object type (the root of the dimension tree) we run through the various functional chains (branches) until we finally reach the last object types (leaves). Along the way we gather all the fact types to eventually group them into a single table based on the identifier for Store (the Store Key). Modeling in this manner will result in denormalized tables containing embedded functional dependencies but we argue there is no need to enforce these since they have been enforced in the operational systems from which the multidimensional data is extracted. Since only the operational tables are used for updating, we believe it is advantageous to model in this way to leverage the performance and comprehension benefits of denormalization.

In our approach, every classification hierarchy level must have a *label* (e.g. City Name) that identifies each level instance. To do this we include the constraint {L} next to the identifying value type for each hierarchy level to explicitly indicate it is the

identifying label for that level. When our model is eventually generated into an OLAP cube, the cube will store this value as the default label in its metadata to unambiguously identify the hierarchy level. As shown in Figure 4-7, we have annotated the schema to indicate cityName as the label for City. Applying a roll-up operation to aggregate measures into the City level of the Store dimension will display the City Name label as we analyze the Cities in which our products are sold.

In addition to multiple and alternative path hierarchies, our approach allows for *shared hierarchies* between dimensions. As illustrated in Figure 4-8, Customer shares the levels city, state, and country as defined in the Store dimension. Using ORM's double-border notation, we depict these objects types and their predicates as *external* to indicate they are imported from another schema in which they are fully defined.

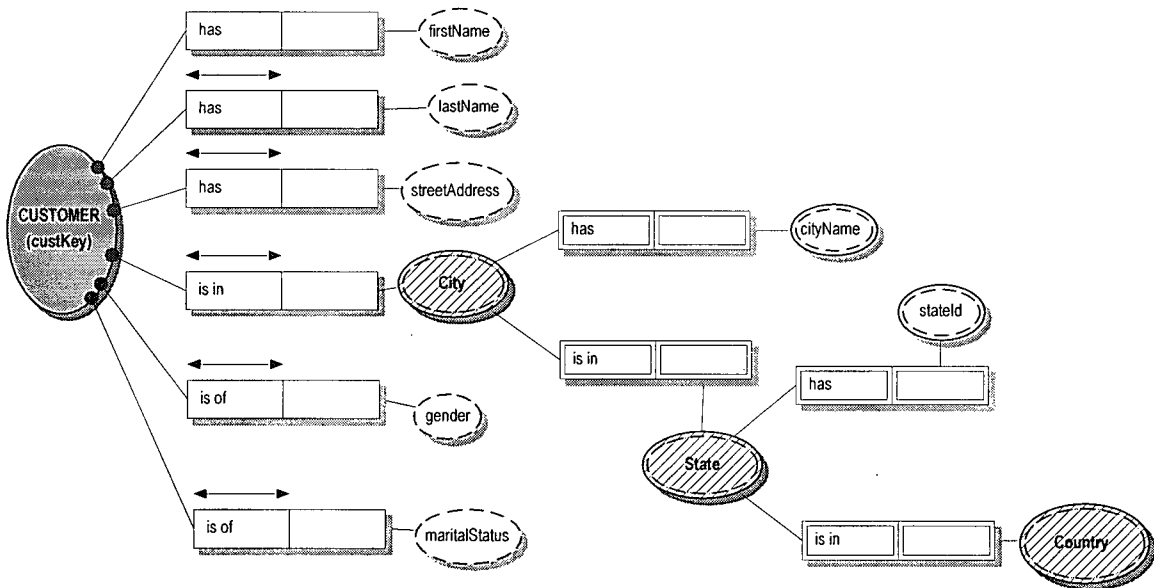


Figure 4-8: Shared Hierarchies in ORM



#### 4.4.3.2. Strictness

As defined in chapter 3, *strictness* means an object at a hierarchy's lower level belongs to only one higher-level object (the target). *Non-strictness* means an object may belong to more than one higher-level object. Our approach uses a combination of cardinality (frequency) and optionality to model the concepts of strictness and non-strictness, as illustrated in the schema in Figure 4-9.

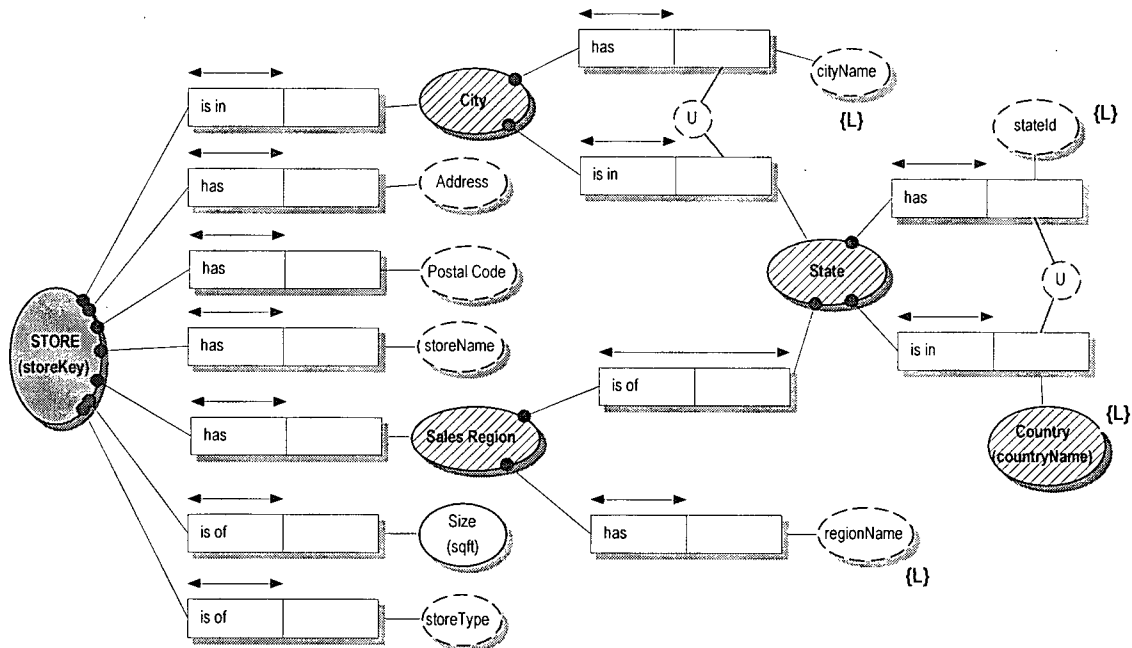


Figure 4-9: Strictness & Non-Strictness in ORM

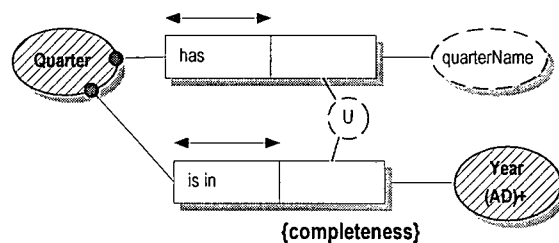
In this Store example, Store and City have a strict relationship because a Store can exist in only one City. To model this strictness, a mandatory constraint on the Store role indicates each store is located in *at least one* city. A many-to-one (n:1) constraint on the first predicate role then indicates each store is located in *at most one* city. Similar strict relationships exist between City and State, with a City existing in only one State.

The Sales Region and State object types form a *non-strict* relationship because a Sales Region can be in more than one State. To model non-strictness, our approach includes mandatory constraints on both roles to indicate that each Sales Region is located in *at least one* State and a State is comprised of *at least one* Sales Region. A many-to-many (*m:n*) uniqueness constraint on the roles then indicates that each Sales Region can relate to *more than one* state. A verbalization of this non-strict relation is:

- *it is possible that a Sales Region is comprised of more than one State and at the same time a State is included in more than one Sales Region*

#### 4.4.3.3. Completeness

As described in chapter 3, *completeness* within a classification hierarchy means that all members belong to one higher-class (target) hierarchy level and that level consists of those members only. To model completeness, we define the {completeness} constraint on the role of the target hierarchy level. We illustrate completeness in Figure 4-10 using the Time dimension from our POS Retail example.



**Figure 4-10: Completeness in ORM**

In Figure 4-10, we have added the constraint {completeness} on the target Year object associated with Quarter. In this “complete” classification hierarchy between Year and Quarter hierarchy levels, all the recorded Quarters form the Year, and all the Quarters that form the Year have been recorded. As for *non-completeness*, our approach assumes all classification hierarchies are non-complete by default.

#### 4.4.3.4. Categorization of Dimensions

Our approach has shown how to model classification hierarchies within dimensions but a multidimensional conceptual model should also consider the *categorization of dimensions* to model additional features of subtypes. To do this we use a generalization-specialization relationship to categorize entities that contain subtypes.

Like ORM, our approach displays subtyping using directed acyclic graphs - a graph of nodes with directed connections, acyclic meaning there are no cycles. An example of categorization using our POS Retail Sales example is shown in Figure 4-11.

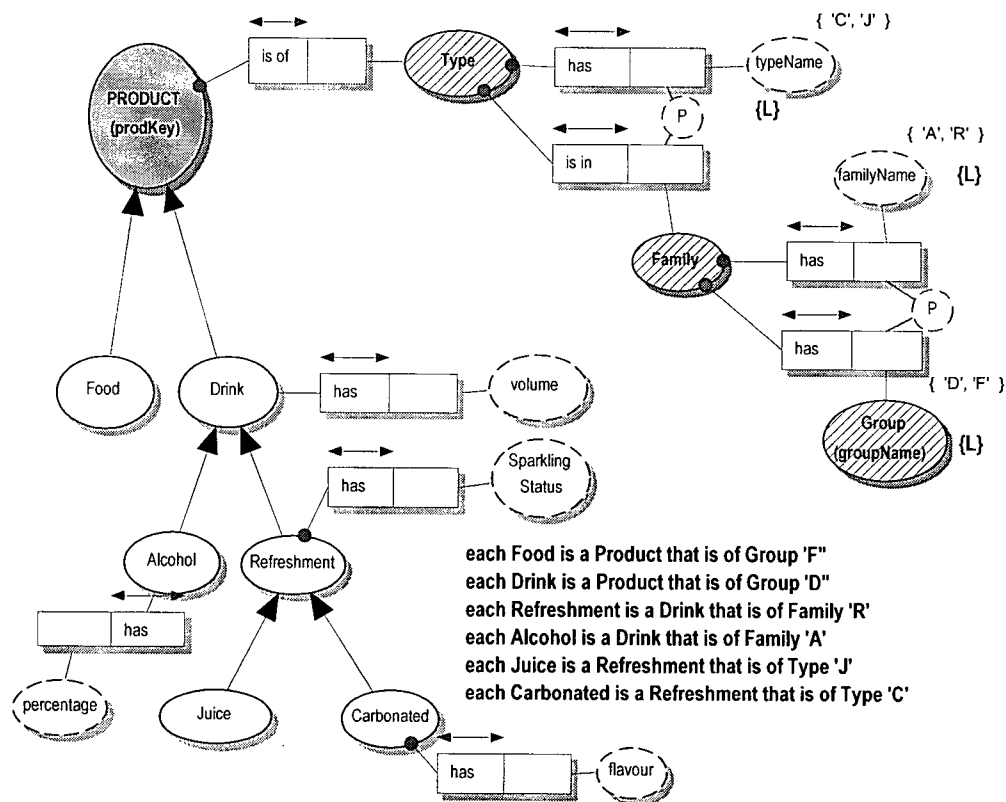


Figure 4-11: Categorization of Dimensions in ORM

The Product dimension contains six subtypes: Food, Drink, Alcohol, Refreshment, Juice and Carbonated. Subtype links are shown as directed line segments from subtypes to supertypes. Subtype nodes are introduced when we have specific roles

for them to play (e.g. Drink has volume). As with ORM, our approach requires formal *subtype definitions* to be declared for all subtypes and written in the diagram. These rules indicate the basis for categorization and must be defined in terms of at least one role played by a subtype's supertype(s). By default, subtypes inherit the identification scheme and all the roles of their supertypes so there is no need to repeat this information.

#### 4.4.3.5. Many-to-Many Relationships Between Events and Dimensions

As described in section 3.4.4.4, we generally consider events to have many-to-one relationships with each dimension. Thus far in our example, we have considered the grain of our sales event to be the individual line item on a sales ticket (e.g. a single product). To illustrate how we represent many-to-many relationships between events and particular dimensions we now assume the grain of interest to be the sales ticket itself. Since there are many line items (e.g. products) per ticket, this means we now have a many-to-many relationship between the product dimension and the sales event. A ticket can consist of more than one product, although each ticket is still purchased in only one store by one customer at one time.

To represent this relationship at the conceptual level our approach includes mandatory constraints on both roles played by the sales event and the product dimension to indicate that each Product is included in *at least one* Sale and a Sale is comprised of *at least one* Product. As shown in Figure 4-12, a many-to-many (*m:n*) uniqueness constraint on both roles then indicates that each Sale can relate to *more than one* Product. A textual rule can be written for this relationship as:

- *it is possible that the same sale (ticket) contains more than one product and at the same time product was part of more than one sale*

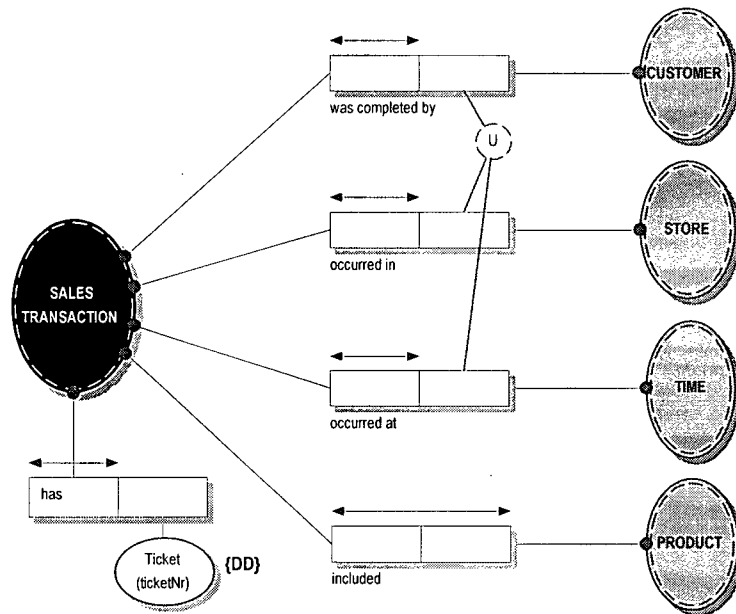


Figure 4-12: Many-to-Many Relationship Between Event & Dimension

#### 4.4.3.6. Degenerate Dimensions

Our approach defines other identifying features of an event, referred to as *degenerate dimensions*, by placing the constraint {DD} next to the identifying object type. In Figure 4-12, we have annotated Ticket, allowing the ticket number to be analyzed in addition to the atomic and derived measures of the sale. This identifying constraint groups individual line items at the ticket level and will prove useful during the generation of our schema into a commercial OLAP tool.

## **4.5. MORM Design Guidelines**

The previous section presented MORM, which reflects our fact-oriented approach to conceptually modeling multidimensional data. We now supplement our approach with several key design guidelines for the development of a multidimensional data model. Since we specialize ORM, the CSDP still applies to the overall design process, however, we consider additional guidelines to incorporate multidimensional concepts.

Based on our experiences developing and utilizing these guidelines in several large scale data warehouse implementations, we believe our guidelines reflect the natural way data modelers and business users understand and view multidimensional modeling. As evidenced by our modeling experiences, this approach is particularly useful for large, complex data warehouses with many events and dimensions. Following our guidelines, modelers are able to systematically develop domain sub-schemas that, together, create the conceptual model for the entire enterprise. The response from business users in our implementations indicates the models created using our guidelines are easily understood.

### **4.5.1. MORM Level 0: Preliminary Segmentation**

The initial phase of our approach “Level 0” involves dividing the universe of discourse into manageable subsections. This allows schema design activities to be divided, where multiple modelers work on models relevant to their domain of expertise. Segmenting the schema in this way creates different levels of abstraction and simplifies the final model. We use ORM’s subschema (submodel) concept to represent the different levels of our MORM models. Using subschemas, our approach is not restricted to using flat diagrams to model large, complex data warehouses.

Level “0” is indicated as such since it is a preliminary step required to complete initial segmentation prior to designing the different levels of the schema. Associated with Level 0 are Guidelines #0a and #0b (shown in Table 4-8), which summarize our overall approach and provide a foundation for the remainder of our guidelines.

#	Guideline
<i>0a</i>	<i>Upon completion of the MORM design process, the multidimensional model will be divided into four levels: business process family definition, business process definition, dimension definition, and event definition.</i>
<i>0b</i>	<i>Before beginning the model, define events and dimensions and indicate shared dimensions and dimensions that share some hierarchy levels.</i>

**Table 4-8: MORM Level 0 Design Guidelines**

Based on Guideline #0a, the multidimensional schema is designed in a top-down fashion by decomposing the model into different levels as outlined in Table 4-9. These levels are discussed further in the sections that follow.

Level	Name	Description
1	Business Process Family Definition	A subschema representing an integrated set of related business processes based on common, shared dimensions.
2	Business Process Definition	A subschema representing a business process and its associated events and dimensions.
3	Event Definition	A subschema representing an event and its associated measures.
4	Dimension Definition	A subschema representing a dimension and its associated hierarchy levels.

**Table 4-9: Four Levels of a MORM Schema**

#### **4.5.2. MORM Level 1: Business Process Family Definition**

Level 1 of our method models a Business Process Family through the use of Event and Dimension object types. This leads us to Guideline #1 as shown in Table 4-10.

#	Guideline
<i>1</i>	<i>Using only Event and Dimension Object Types, draw a subschema representing all business processes considered.</i>
<i>2</i>	<i>Define instances of all fact types (objects and their predicates) as external to indicate that definitions of event and dimension objects and their roles exist in subsequent levels.</i>

**Table 4-10: MORM Level 1 Design Guidelines**

Figure 4-13 shows the first level of a model representing the family of business processes from our case study.

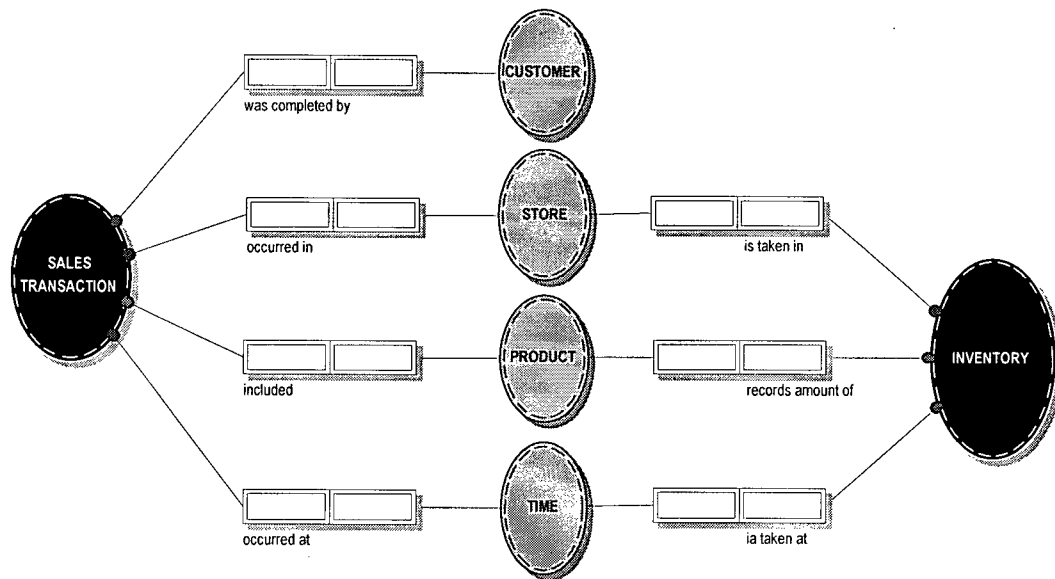


Figure 4-13: MORM Level 1 - Retail Business Process Family

Event object types represent the Sales Transaction and Inventory events while Dimension object types represent the Time, Store, Customer, and Product dimensions. Dimension objects with roles spanning two event objects at this level indicates the business processes share that dimension. At this level, the predicates and objects of all fact types are annotated with double-border ellipses to indicate they are external. This leads us to Guideline #2 of our approach shown above in Table 4-10.

Using the external property in this way allows us to reference the event and dimension objects that will be defined in another schema level in subsequent phases of the design process. Once we define the dimensions, all events can use them without having to define them again. This ensures the integrity of our data model by allowing us to define object types and their roles, and then refer to these definitions throughout other subschemas within the entire data model.



#### 4.5.3. MORM Level 2: Business Process Definition

Level 2 of our approach involves drawing a subschema for each business process considered. Shown in Table 4-11, Guidelines #3 and #4 guide the design at this level.

#	Guideline
3	<i>Draw a subschema representing a single business process using a single event object and its associated dimension object types and predicates.</i>
4	<i>Annotate instances of all event and dimension object types as external, however, fully define roles (predicates) between each object.</i>

Table 4-11: MORM Level 2 Design Guidelines

Figure 4-14 shows the POS Retail Sales business process modeled using our approach. As in Level 1, object types are annotated as external to indicate they are defined elsewhere in the model. However, detail is introduced at this level for the roles between the event and its associated dimensions.

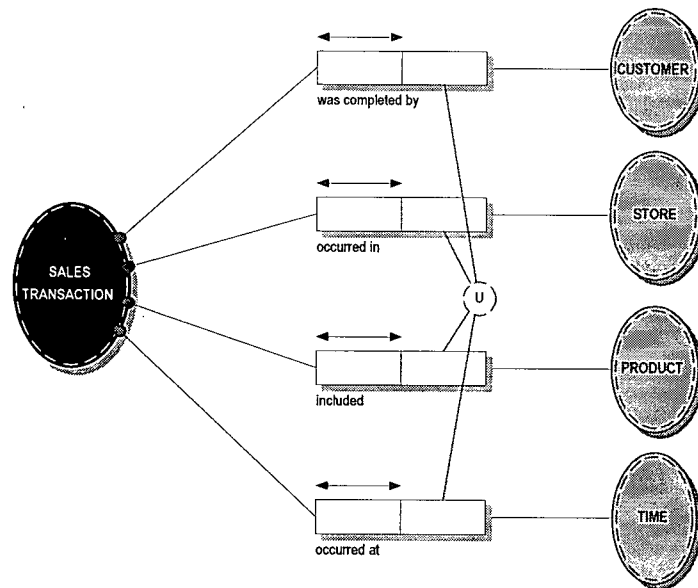


Figure 4-14: MORM Level 2 - POS Retail Sales Business Process

In our example, all roles played by the event object are mandatory (e.g. Sale must have *at least one* Store), thus are explicitly shown by a mandatory role dot where a role connects with the object type. Internal uniqueness constraints over the roles indicate each

object playing that role does so only once (e.g. each Sales Transaction occurs in *at most one* Store). The external uniqueness constraint across the four roles indicates each Sale occurs for *at most one* Store, Customer, Product, and Time combination.

#### 4.5.4. MORM Level 3: Event Definition

Level 3 of our MORM method creates a subschema for all the measures of interest in a business process. Shown in Table 4-12, Guidelines #5, #6, and #7 of our approach guide subschema development throughout this level.

#	Guideline
5	Draw a subschema defining an event and all relevant measures of the business process.
6	Fully define the event object and other objects for each of the measures considered; define derivation rules for any derived measures.
7	Define roles between the event object and each of its associated measure objects.

Table 4-12: MORM Level 3 Design Guidelines

Figure 4-15 shows a Level 3 schema representing event measures from our case study.

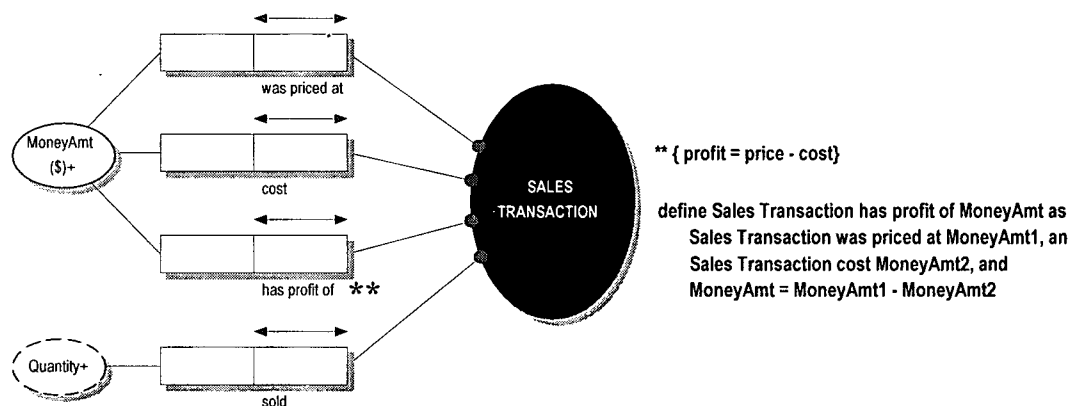


Figure 4-15: MORM Level 3 - Sales Transaction Event

Since the Level 3 subschema is the original source of the event object definition, the object is drawn with a single border. Measure objects are defined and rules are developed to reflect additivity and indicate derived and stored measures. Rules are defined for all objects by indicating mandatory roles and uniqueness constraints for each of the measures.

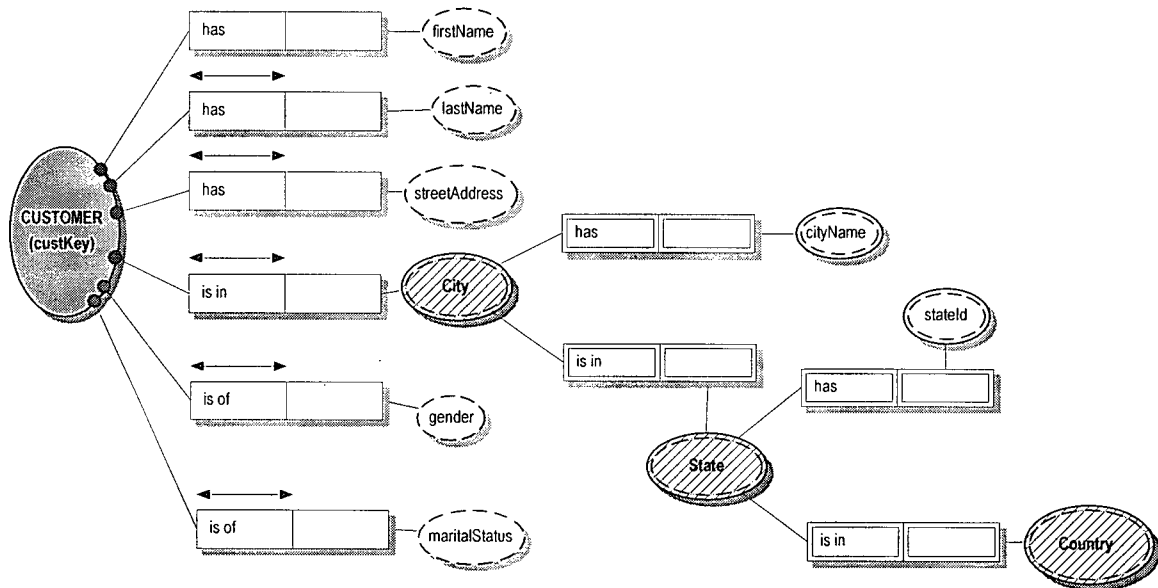
#### 4.5.5. **MORM Level 4: Dimension Definition**

Level 4 of our approach models dimension content using dimension objects as the root of dimension trees and hierarchy object types to represent the hierarchy levels within the dimension. This leads to Guidelines #8, #9, #10 and #11 of our approach shown in Table 4-13.

#	Guideline
8	<i>Draw a subschema representing each dimension of the business process.</i>
9	<i>Draw a dimension object for the dimension and hierarchy objects for each of its hierarchy levels, define roles played by each.</i>
10	<i>If a dimension or hierarchy level has been previously defined, draw its objects and predicates and annotate them as external (i.e. do not define a dimension or hierarchy level twice).</i>
11	<i>Define objects and roles for each of the remaining dimension attributes.</i>

**Table 4-13: MORM Level 4 Design Guidelines**

Figure 4-16 shows a Level 4 model representing the Customer dimension, its different hierarchy levels (e.g. City, State, and Country) and other dimension information. At this level, a dimension object forms the root of a dimension tree, where each node is an object type and each edge is a functional (n:1 or 1:1) predicate.



**Figure 4-16: ORM Level 4 - Store Dimension**

Hierarchy levels are indicated with hierarchy object types, while external hierarchy objects and predicates represent hierarchy levels defined in another source subschema and shared by this dimension. For example, the external fact types for City, State, and Country are defined elsewhere (e.g. the Store dimension) and annotated here as external to indicate they are shared with that dimension.

It is important to note dimensions that share hierarchy levels do not need to share the whole hierarchy. For example, the address hierarchy of the Store dimension could just include the City and State levels if required.

#### **4.5.6. Design Guideline Summary**

Having described each of the design levels and guidelines separately throughout the previous sections, we now summarize the entire ORM design process in Table 4-14.

#	Level	Guideline
0a	0	<i>Upon completion of the MORM design process, the multidimensional model will be divided into four levels: business process family definition, business process definition, dimension definition, and event definition.</i>
0b	0	<i>Before beginning the model, define events and dimensions and indicate shared dimensions and dimensions that share some hierarchy levels.</i>
1	1	<i>Using only Event and Dimension Object Types, draw a subschema representing all the business processes considered</i>
2	1	<i>Define instances of all fact types (objects and their predicates) as external to indicate that definitions of event and dimension objects and their roles exist in subsequent levels.</i>
3	2	<i>Draw a subschema representing a single business process using a single event object and its associated dimension objects and predicates.</i>
4	2	<i>Annotate instances of all event and dimension object types as external, however, fully define roles (predicates) between each object.</i>
5	3	<i>Draw a subschema defining an event and all relevant measures of the business process.</i>
6	3	<i>Fully define the event object and measure objects for each of the measures considered; define derivation rules for any derived measures.</i>
7	3	<i>Define roles between the event object and each of its associated measure objects.</i>
8	4	<i>Draw a subschema representing each dimension of the business process.</i>
9	4	<i>Draw a dimension object for the dimension and hierarchy objects for each of its hierarchy levels, define roles played by each.</i>
10	4	<i>If a dimension or hierarchy level has been previously defined, draw its objects and predicates and annotate them as external (i.e. do not define a dimension or hierarchy level twice)</i>
11	4	<i>Define objects and roles for each of the remaining dimension attributes.</i>

Table 4-14: MORM Design Guideline Summary

## 4.6. An Evaluation of MORM

Through a specialization of Object Role Modeling, we have proposed a natural and expressive model that represents the structural properties of multidimensional data at the conceptual level. We believe our fact-oriented approach, as exemplified by MORM, provides many benefits over other related multidimensional models.

To the best of our knowledge, we have presented the first fact-oriented approach to conceptual multidimensional modeling. In doing so, we leverage the fact-oriented paradigm and introduce several new multidimensional constructs to ORM – the *Event Object Type*, *Dimension Object Type*, and *Hierarchy Object Type*. We take the concepts and basic ideas of the multidimensional view of data and propose an approach based on the fact-oriented paradigm to model multidimensional data at the conceptual level. We

believe this utilization of the fact-oriented paradigm provides us with a conceptual multidimensional model that is more natural and simpler than existing models. As such, MORM provides a solid basis for solving conceptual multidimensional modeling problems with a more intuitive and natural conceptual model than existing approaches.

We propose MORM as a specialization of ORM model by defining additional graphical constructs and guidelines to consider the characteristics of multidimensional modeling. Our technique allows us to consider key multidimensional properties at the conceptual level, providing semantics that distinguish qualifying (dimension) and quantifying (event) data. Other key multidimensional properties supported by our approach include multiple and alternative path classification hierarchies, strictness and completeness, many-to-many relationships between events and dimensions, additivity, derived and atomic measures, and the categorization of dimensions.

Based on our practical experience, we have also provided design guidelines to properly and easily apply MORM. We believe these guidelines reflect the natural way users and data modelers think about multidimensional data and lead us to a very simple yet powerful multidimensional model. Through our guidelines, we have shown how MORM subschemas can be successfully used for multidimensional modeling at four levels of complexity – business process family, business process, event and dimension. Our multilevel subschemas group different levels of abstraction to simplify conceptual design when modeling large and complex data warehouses.

A significant advantage of our approach is that it uses a widely accepted fact-oriented modeling language. By basing our approach on the established ORM model we

enable the transfer of research results published in the context of ORM. As such, we can apply previously discussed evaluation results (Halpin & Bloesch, 1999) to our approach.

By specializing ORM, we also minimize the effort required of data modelers to learn new modeling notations and methodologies for data warehouses and OLAP applications. This way, we ensure a shallow learning curve since data modelers can combine MORM elements with classical ORM elements and, although the approaches will be different, conceptual data models for OLTP and OLAP applications may be specified using a uniform notation.

#### **4.7. Summary**

In this chapter we introduced MORM, our fact-oriented multidimensional modeling approach which introduces multidimensional constructs to ORM. We have demonstrated how our approach handles basic and advanced multidimensional concepts and have shown how our MORM guidelines are used for successful multidimensional modeling at various levels of complexity. Based on our experience, our guidelines provide various levels of abstraction and simplify conceptual design when modeling large data warehouses. Finally, we evaluated our model and discussed its strengths with respect to multidimensional concepts conceptual modeling language criteria. Among other benefits, we have shown that our approach provides a natural, yet powerful way to model multidimensional data and allows domain experts to validate the model in terms of sentences and sample data populations. As we will demonstrate in the next chapter, another significant benefit of our model is that can be automatically mapped to logical and physical schemas and implemented using existing technologies.

## 5. APPLYING MORM: A CASE STUDY

### 5.1. Introduction

This chapter describes the application of our fact-oriented modeling approach throughout the *multidimensional modeling implementation lifecycle*, which we define to include the four phases outlined in Table 5-1.

Phase	Description
1	Creating a <b>conceptual schema</b> in a graphical modeling tool.
2	Mapping a <b>logical schema</b> from the conceptual schema.
3	Generating a <b>physical schema</b> from the logical schema.
4	Building an <b>OLAP cube</b> from the physical schema.

Table 5-1: Multidimensional Modeling Implementation Lifecycle

To test the practicality and usability of our approach we demonstrate how our model can be implemented throughout this lifecycle using existing technologies. In doing so, we first provide an overview and rationale for our selected development environment tools, then present the implementation details for each phase. Following the implementation we evaluate our results and identify the experiences we have learned from our case study.

### 5.2. Development Tools

Before describing the implementation details of our approach, we first introduce our chosen development tools. These tools are required to achieve various tasks associated with the four lifecycle phases of our implementation. The following sections review our selections for tools to facilitate conceptual and logical modeling, relational database storage, and OLAP cube generation.



### **5.2.1. Conceptual & Logical Modeling Tool: VisioModeler™**

ORM is supported by a variety of modeling tools, including Microsoft® *VisioModeler*, Microsoft® *Visio 2000 Enterprise*, and Microsoft® *Visual Studio.NET*. Formerly known as InfoModeler, VisioModeler was renamed when Visio Corporation acquired InfoModeler in 1997. Visio then rewrote the VisioModeler tool to use the Visio drawing engine and released the first version of the Visio Modeling Engine add-in with Visio 2000 Enterprise. With the subsequent acquisition of Visio in 2000, Microsoft released VisioModeler as unsupported product.

The Visio 2000 Enterprise tool supports updated drivers and diagramming for most of the ORM constructs, however, relational mapping is not supported. Microsoft's second, more advanced version of the Visio Modeling Engine is found in Visual Studio Enterprise Architect (VSEA), released in April 2002 as part of Visual Studio.NET. VSEA provides the most current support for ORM modeling with many improvements to diagramming and relational database mapping, however, it is quite expensive and unavailable on trial basis.

Although VisioModeler is a discontinued product with outdated database driver support, we chose it as our modeling tool because of its functionality and availability. It may be unsupported, but VisioModeler remains a feature rich, mature modeling tool that allows the creation of ORM models and subsequent mapping to a wide range of database systems. VisioModeler is easily accessible as a free download from Microsoft® Corporation ([www.microsoft.com](http://www.microsoft.com)), whereas other ORM modeling tools are cost prohibitive.

We use VisioModeler to formalize our database design by working at the conceptual level using natural-language facts, verifying our design using real-world

example data, validating and mapping a logical model, and finally producing a physical schema using 32-bit ODBC drivers. VisioModeler models can be automatically mapped to database schemas for implementation on most popular relational databases. To do this, VisioModeler automatically generates table diagrams (i.e. a logical model) that can be automatically translated into SQL code and applied to the database system of choice.

### **5.2.2. Relational Database: Microsoft® SQL Server™ 2000**

We chose Microsoft® SQL Server™ 2000 (SQL Server) as our relational database management system (RDBMS) because of its market share and availability. SQL Server is a family of database products appropriate for a broad range of solutions, including small and large business applications, e-commerce, and data warehousing. Marketed by Microsoft as a “complete database and analysis product” ([www.microsoft.com](http://www.microsoft.com)), SQL Server meets the storage requirements of large businesses yet provides easy-to-use data storage services to individuals and small businesses.

Of the eight versions available, we chose Microsoft® SQL Server™ 2000 Enterprise Evaluation Edition. This edition is a full-featured version available as a download from Microsoft ([shop.microsoft.com/devtools](http://shop.microsoft.com/devtools)) for a minimal shipping and handling fee. Intended only for feature evaluation, this is a 120-day time-limited version of SQL Server 2000 Enterprise Edition licensed for demonstration, testing, examination, and evaluation. SQL Server is attractive not only because we can evaluate the complete set of data management and analysis features without purchasing the full version, we can also install it on the desktop without running a server based operating system.

### **5.2.3. OLAP Tool: Microsoft® SQL Server™ Analysis Services**

Bundled as a component of SQL Server 2000, Analysis Services is a multidimensional analysis tool with OLAP and data mining capabilities. A logical choice for us because of its integration with SQL Server, Analysis Services can also extract data from the data warehouses and data marts of many other data sources. Its data can be stored multidimensionally within relational databases (ROLAP), as separate, high-performance multidimensional data structures (MOLAP), or hybrid combinations of both (HOLAP). Through its multidimensional cubes, Analysis Services allows us to turn Grocery data stored in the star schema event and dimension tables of our SQL Server database into meaningful, easy-to-navigate business information.

### **5.3. STEP 1: Creating the Conceptual MORM Schema**

Having described our example case study and the development tools we'll use for our implementation, we now begin the conceptual design of our multidimensional model. We use VisioModeler as our modeling tool to examine and describe the application domain in a way that is clear and easy to understand. A multidimensional model representing the MORM constructs described in the previous chapter should be easily designed in the VisioModeler tool. Our output at this step is a conceptual model consisting of natural language facts and intuitive diagrams that serves as a key communication tool between the end user and designer.

We demonstrate the practicality and feasibility of our MORM model through our Retail POS example presented in the previous chapter. Since we have revised this example in several places, we restate it here to avoid confusion.

*A large grocery chain with 100 stores is spread over a five-state area. Each of the stores has a full complement of departments, including grocery, frozen foods, dairy, meat, produce, bakery, floral and health/beauty aids. Each store has roughly 60,000 individual products on the shelves, each with bar codes referred to as stock keeping units (SKUs). As customers purchase products at the cash register, sales data is gathered by scanning bar codes into a point of sales (POS) system.*

*Management is interested in understanding customer purchases as captured by the POS system and they have decided to analyze the POS Retail Sales process. They hope to understand which products are selling to which customers at which stores during which times.*

*Optimized inventory levels in our grocery stores can have a major impact on chain profitability. Making sure the right product is in the right store at the right time minimizes out-of-stock situations and reduces overall inventory carrying costs. To better understand the inventory-sales relationship, management would also like the ability to analyze daily quantity on hand inventory levels by product and store.*

### **5.3.1. VisioModeler Diagram Workspace**

Before we discuss the design of our conceptual models, we provide a brief overview of the VisioModeler Diagram Workspace used to create and manage basic data modeling tasks. Using toolbars and editors in the main VisioModeler window we create the conceptual models of our multidimensional database in an object-role modeling document using our MORM modeling approach. Each conceptual model is specified in an ORM Modeling Diagram (.IMO file), with a set of graphic symbols and specialized tools used to design our data model.

For all of our submodels, we first begin conceptual design of our application domain in an ORM modeling diagram, then build and refine the mapped logical model in a dictionary document. A modeling document (.IMO file) is saved as a file that can be

opened and closed like any other file, however, a dictionary document (.IMD file) is associated with a particular modeling document or project and can only be opened when its associated modeling document or project is open. As illustrated in Figure 5-1, a model and its associated dictionary comprise the specification of the model.

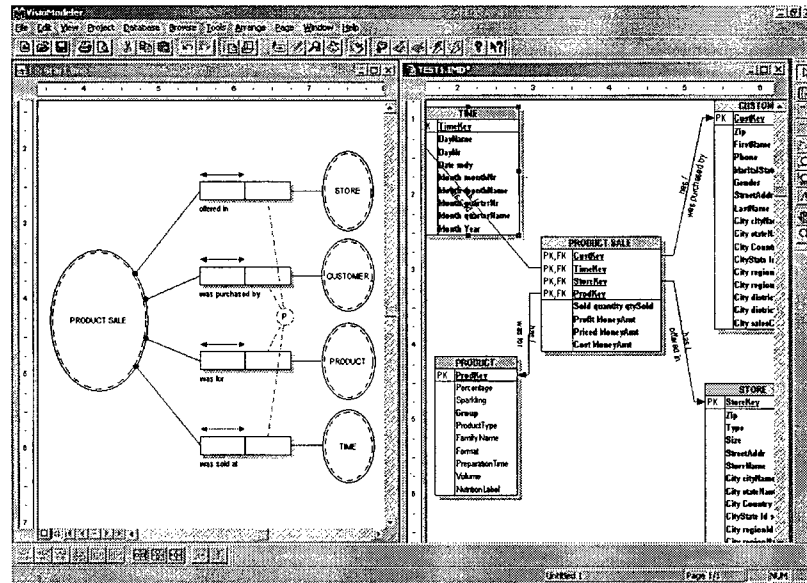
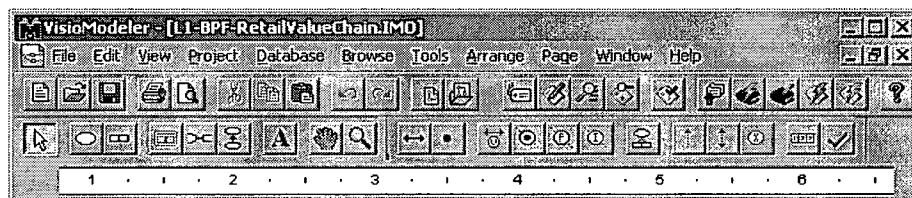


Figure 5-1: MORM Model and Associated Dictionary Document

VisioModeler provides several ways to create and edit the symbolic components of fact types in our modeling diagram. Primarily, the *Tools* palette is used to draw and connect object types and predicates one by one and the *Constraints* palette is then used to



add constraints to the diagram. Both Palettes are shown in Figure 5-2.

Figure 5-2: VisioModeler Tool and Constraint Palettes

The *Fact Editor* is also used to create and edit a fact type in our ORM model. Shown in Figure 5-3, the Fact Editor greatly simplifies the entry of facts by automatically converting the entered text to the appropriate symbols in the ORM diagram. The Editor checks the syntax of a fact type and as shown in Figure 5-4, verifies the correctness of a fact type's constraints using example data.

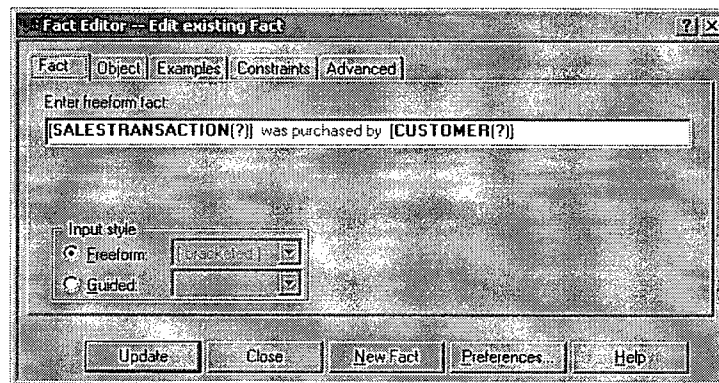


Figure 5-3: VisioModeler Fact Editor Window

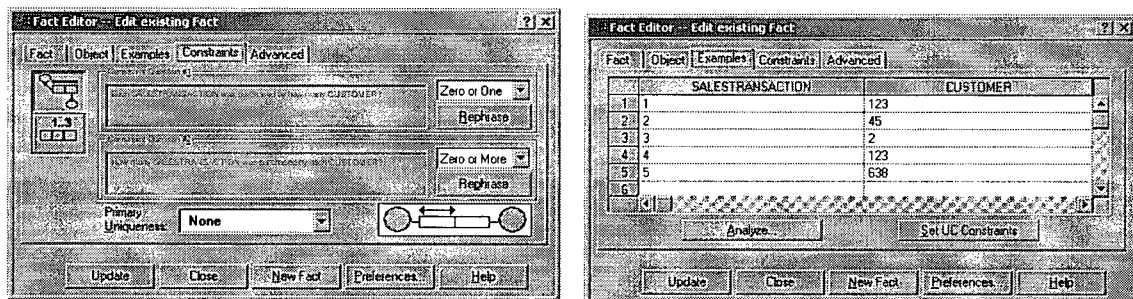


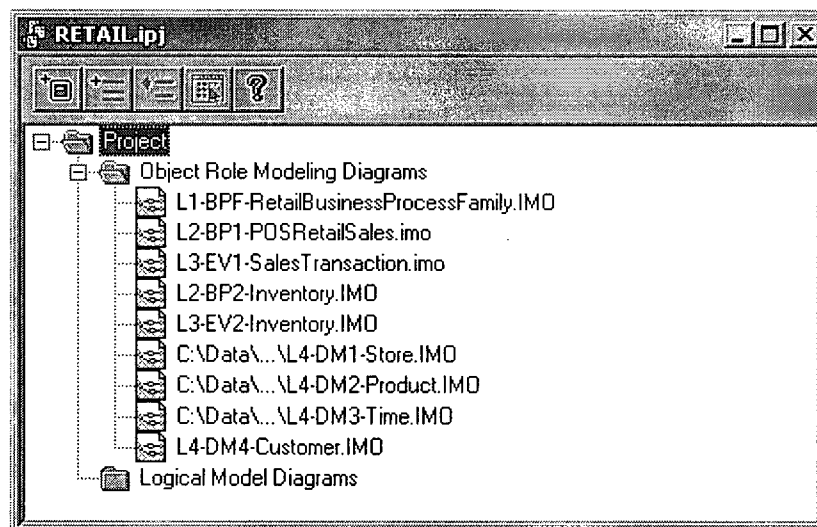
Figure 5-4: Fact Editor Constraints and Associated Data Examples

### 5.3.2. Creating a ORM Project

We use VisioModeler's project-based development feature to support our multi-level design approach to multidimensional modeling. A *Project* is created as a set of model documents containing various ORM subschemas that make up the specification of our entire multidimensional data model. Within a Project, we subdivide a complex multidimensional model into smaller, manageable submodels associated with the design

levels outlined in our approach. These submodels use multiple modeling documents and can be developed by different modelers with different domain expertise and subsequently re-used across the data warehouse.

Shown in Figure 5-5, our VisioModeler Project shows how we have defined and organized our multidimensional model using multiple source documents. Our submodels are organized using a tree diagram and are categorized by type of model document. With the creation of our Project, VisioModeler has generated a Project file (.IPJ) and a directory to store all the models associated with our Project. Our files are named to indicate design level (e.g. L1, L2) and Type (e.g. EV for Event, DM for Dimension).



**Figure 5-5: VisioModeler's Project Window**

When building its dictionary, VisioModeler combines the contents of the model documents listed in our Project window to form an integrated model and saves this information in a dictionary document. In building the dictionary, VisioModeler merges our Project files to form a complete, mapped model of our multidimensional application domain and saves this information in a dictionary (.IMD) file. The build process checks

and validates overlapping model components to ensure model integrity. Defining a project in this fashion supports MORM Guideline #0a, which states that upon completion of the MORM design process, our model should be divided into four levels.

### 5.3.3. Creating MORM Schemas

Having described our Project and the fundamentals of the VisioModeler workspace, we next discuss the process of creating individual submodels. Consistent with our approach and the design guidelines presented in chapter 4, we use the VisioModeler workspace to create subschemas for each of our design levels. As a prerequisite step to schema development we follow Guideline #0b and define events, dimensions, hierarchies and hierarchy levels for each of our business processes. The resulting segmentation for the Retail Sales business process is shown in Table 5-2.

Business Process	Event	Measure	Dimension	Dimension Hierarchies	
				Level	Shared
POS Retail Sales	Sales Transaction	Price Cost Profit Ticket # Quantity	Store	City	Y
				State	Y
				Country	Y
			Product	Group	N
				Family	N
				Type	N
				Band	N
			Customer	City	Y (Store)
				State	Y (Store)
				Country	Y (Store)
			Time	Month	N
				Quarter	N
				Year	N

Table 5-2: MORM Design Guideline #0b

With the preliminary segmentation activities of Level 0 addressed in Table 5-2, we now complete schemas for Levels 1 through 4, for our business process family, business processes, events, and dimensions. Since we have demonstrated all multidimensional modeling aspects of our approach through examples in the previous



chapter, we will not reiterate development details of our case study schemas in the body of this chapter. Instead, we include complete VisioModeler subschemas for each level of our case study in Appendix B for the reader's reference.

Throughout the development of each of our schemas, we followed our MORM design guidelines to ensure multidimensional concepts were accurately represented, while adhering to ORM's CSDP for general ORM design principles and steps. Consistent with our approach, familiar information examples were first developed for our case study. Those examples were then translated into elementary facts and conceptual schemas showing all the fact types were drawn for each submodel. To support the multidimensionality inherent in our Retail Sales data, our schemas make extensive use of our proposed MORM constructs - Event, Dimension, and Hierarchy Level Object types.

Upon completion of the MORM design process, our resultant multidimensional model is divided into four levels. Before the final integration of these levels, we validate our subschemas using VisioModeler's **CheckDocument** option from the main toolbar. This function checks to see if our subschema is valid (e.g. no contradictory constraints) and helps us refine our model and correct any errors prior to logical mapping.

#### **5.4. STEP 2: Mapping the Logical Schema**

Having checked and validated our integrated source model documents, we now build the data dictionary and map the conceptual model to the logical model. To do this we use the **BuildDictionary** option from the main toolbar. VisioModeler builds a data dictionary using the specifications designed in our source model documents, then validates and maps the conceptual schema in the dictionary to a logical schema. This

section briefly explains the process of building the dictionary file, validating the conceptual model, correcting errors/warnings and mapping the logical model.

#### 5.4.1. Building the Data Dictionary

VisioModeler builds a data dictionary based on the contents of our project and saves the dictionary document as a .IMD file that acts as a central repository for essential information about our integrated model. The data dictionary contains complete information about the components of our model, including a factbase (the facts that describe our application domain), the conceptual schema, mapping paths, and the mapped logical schema. When building the dictionary, VisioModeler gathers information in our source model documents, consolidates the models associated with our project into one dictionary, forms a conceptual model in the dictionary, validates this model, and then maps the validated model to a logical model.

VisioModeler's Output window identifies and locates any modeling errors in our model and dictionary documents. The Output window displays information, progress notes, warnings; and error messages found during many VisioModeler operations, including building the dictionary and validating a model. Our generation results are shown in the Output window in Figure 5-6.

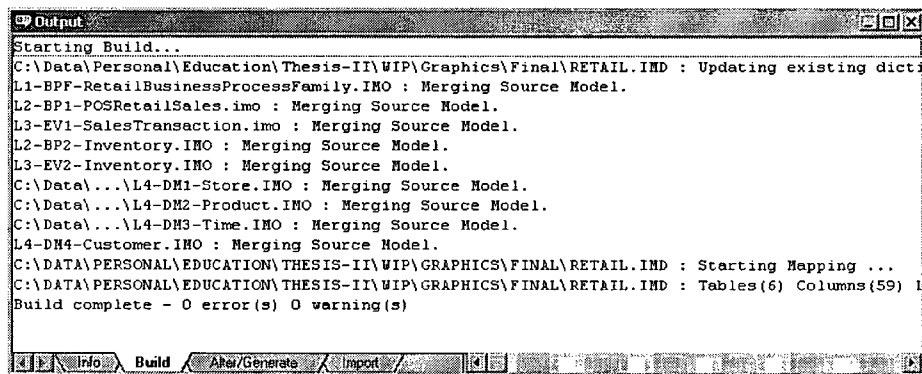


Figure 5-6: Output Window Showing Build Results

#### **5.4.2. Relational Mapping (Rmap) Procedure**

As part of the dictionary generation process, VisioModeler uses Rmap, an algorithm used to group our fact types into tables. The complete version of Rmap includes details for completely mapping all graphical constraints, however, an exhaustive treatment of the full procedure is beyond the scope of this thesis. We introduce the procedure here to provide context for our mapping step and refer the reader to Ritson and Halpin (1993) for detailed coverage of the procedure.

Rmap guarantees a redundancy-free relational design and restricts the number of tables, ensuring each fact type maps to only one table in such a way that its instances appear only once. If the conceptual fact types are elementary, then the mapping is guaranteed to be free of redundancy since each fact type is grouped into only one table, and fact types which map to the same table all have uniqueness constraints based on the same attribute(s). To achieve this Rmap uses two basic rules:

- 1. Fact types with compound uniqueness constraints map to separate tables.*
- 2. Fact types with functional roles attached to the same object type are grouped into the same table, keyed on the object type's identifier.*

While ORM describes facts in terms of simple sentences, relational schemas describe the world in terms of tables with attributes. Through Rmap, the fact types in our ORM model map to the tables in Figure 5-7, which depicts the logical database diagram generated from our model.

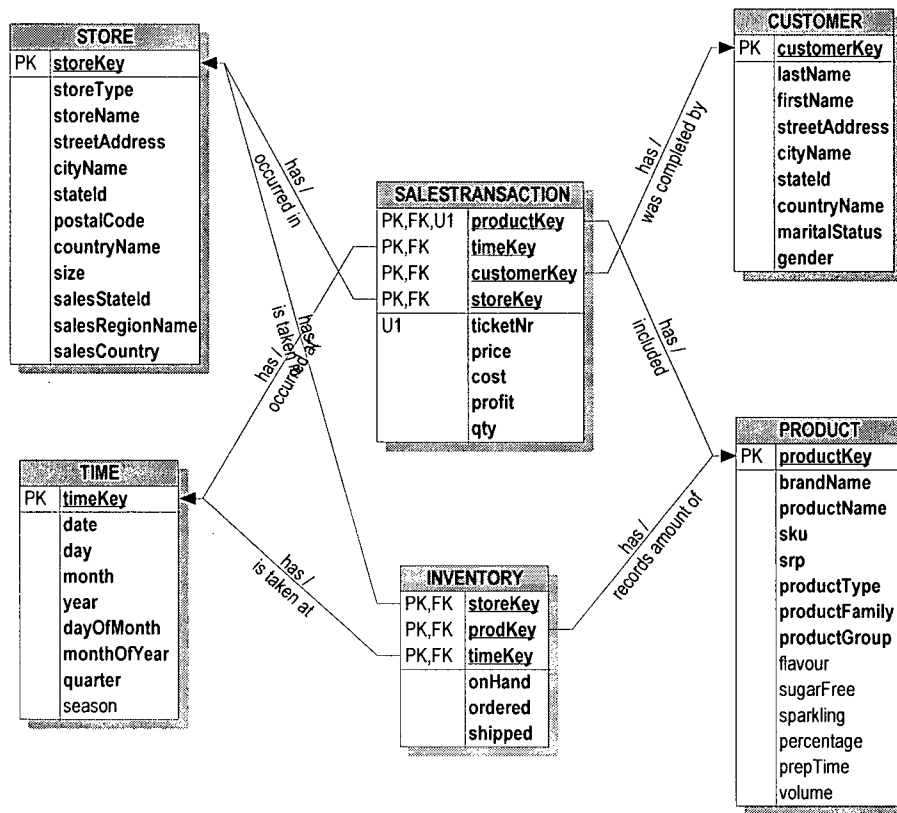


Figure 5-7: Logical Model Mapped From ORM Schema

Using our MORM modeling approach, the resulting logical model essentially consists of two denormalized star schemas for our POS Retail Sales and Inventory business processes. The Sales schema is composed of a central Sales Transaction table linked by foreign key connections to the Store, Product, Time, and Customer dimension tables. At the center of the Inventory schema is a central Inventory table linked to Store, Product and Time. Uniqueness constraints are mapped to *primary key* (PK) or unique *constraints* (U) and primary keys are underlined. A mandatory role constraint is indicated with a *bold* attribute and its rules are enforced in the Data Definition Language (DDL) script generated for the physical database in the next step of our lifecycle process.

### **5.4.3. Editing the Logical Model**

After we build the data dictionary, VisioModeler allows us to edit and refine our logical tables, which are part of the logical model contained in our dictionary. Since some automatically generated names may not be ideal, we modify the resulting tables and map technical column and table names to more meaningful names.

We edit our mapped logical schema using VisioModeler's **EditDictionary** function. This executes a build that opens the dictionary window as a workspace for viewing and refining the mapped logical schema. The Logical Tools palette and the Table/Entity Selector are then used to create, edit, and manage schema.

An attractive feature of VisioModeler throughout this step is its Window option, which allows us to switch viewing windows between the ORM window (.IMO file), our dictionary/logical window (.IMD file), and the Output window containing our generation messages.

## **5.5. STEP 3: Generating a Physical Schema**

Once we have edited and validated our logical model in the dictionary, we generate our physical schema in SQL Server 2000, our selected DBMS. This schema will serve as the underlying data warehouse for storing our relational star schema data. The remainder of this section briefly explains how we use VisioModeler drivers in conjunction with 32-bit ODBC drivers to generate a new physical database schema by connecting to and exchanging information with our SQL Server relational database. We include information about generation options, database connections, and target databases.

### 5.5.1. Schema Generation Options

Through VisioModeler's **GenerateDatabase** function, we run the Generate Wizard (shown in Figure 5-8) to lead us through the physical schema generation process.

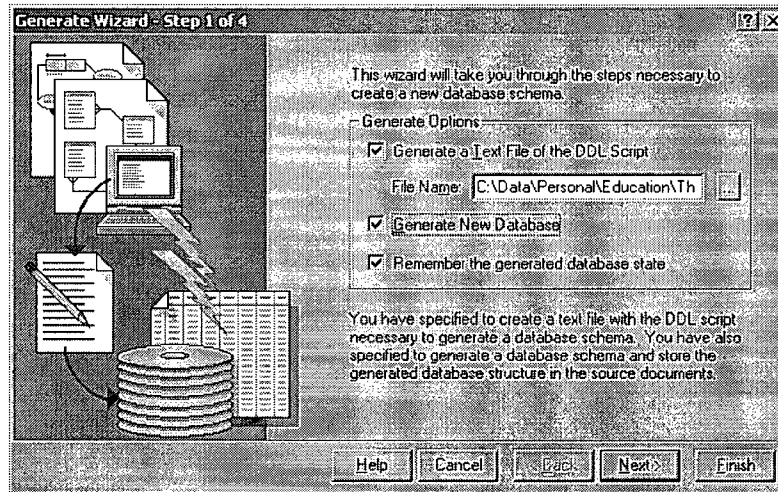
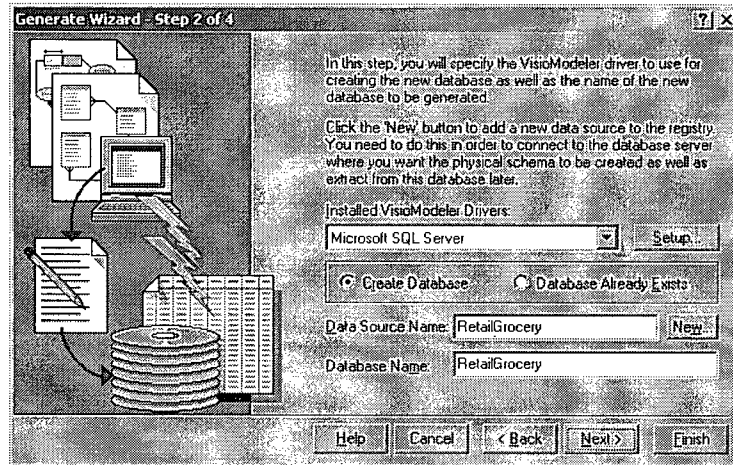


Figure 5-8: Options Within VisioModeler's Generate Wizard

Based on the logical model in the dictionary, VisioModeler provides two ways for us to generate our physical database schema: (1) using a DDL script or (2) directly through a 32-bit ODBC connection. We create our physical database by connecting directly to SQL Server through an ODBC connection, but we also generate a DDL script for reference purposes.

### 5.5.2. Generating Directly Through ODBC

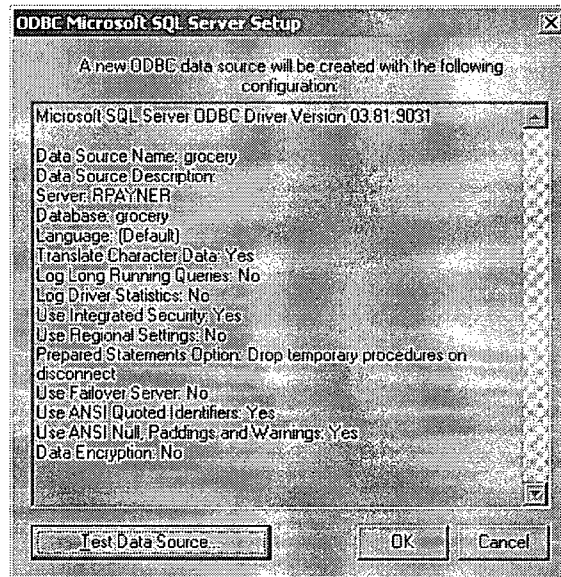
Before we generate our physical schema we must complete several prerequisite system configuration tasks to ensure we have properly installed and configured SQL Server 2000. These tasks ensure we have the necessary client software and adequate access rights. Upon successful completion, we proceed with the SQL Server connection as shown in Figure 5-9.



**Figure 5-9: Associating an ODBC driver with a VisioModeler Driver**

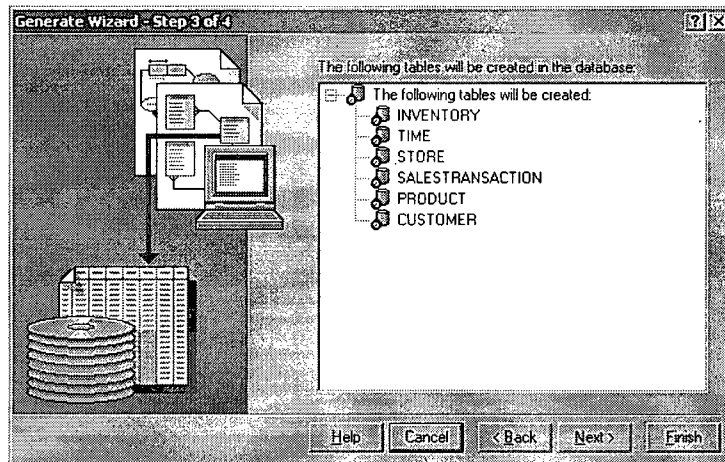
In choosing driver and database options we first select the SQL Server driver we wish to use from the list of available drivers and choose the generate options for our SQL database. The SQL Server driver tells VisioModeler what kind of script to generate, how to map constraints, and how to specify advanced features for our database application. VisioModeler uses this information to extract a physical catalog, synchronize a logical schema, generate a physical schema, and alter a physical schema.

After configuring our driver we create a data source for our Retail Grocery database using Windows ODBC Administrator. This data source references our SQL Server Grocery database and includes the data we will access as well as the information essential to access that data, such as the name of the database, the server on which it resides, and the network information. We associate the SQL VisioModeler driver with a 32-bit ODBC driver to communicate with SQL Server then select the chosen data source. Upon providing a username and password in the connect dialogue box we successfully establish a connection to our data source, as shown in Figure 5-10.



**Figure 5-10: ODBC Data Source Definition**

After successfully connecting to our SQL Server data source, our final task before schema generation is previewing the tables VisioModeler will add to our physical database (see Figure 5-11). After reviewing these for accuracy, we proceed with the generation process and create our physical Grocery tables in SQL Server.

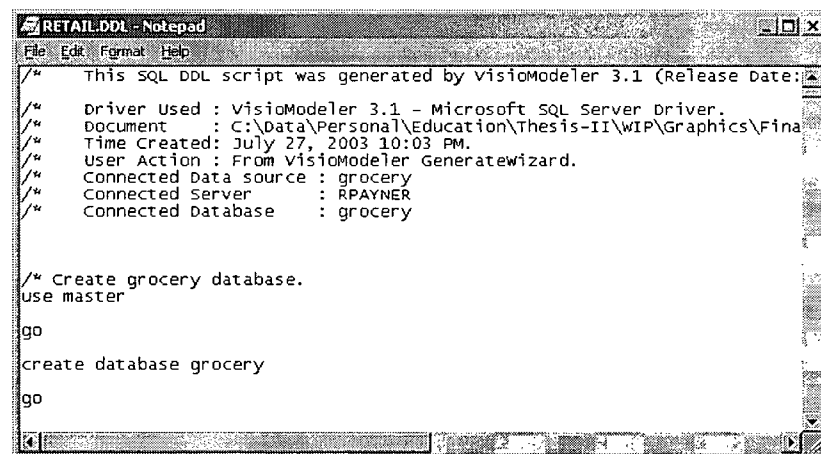


**Figure 5-11: Table Preview in VisioModeler Generate Wizard**



### 5.5.3. Generating a DDL Script

For reference purposes, we also instruct VisioModeler to generate a data definition language script based on the logical model in its dictionary file. This script can be used to create, modify, and delete our database and its tables, columns, rules, and indexes. We can save the DDL script as a text file to review, modify, and run from our SQL Server DBMS if desired. A snapshot of the generated DDL file is shown in Figure 5-12.



```
/* This SQL DDL script was generated by VisioModeler 3.1 (Release Date:
/* Driver Used : VisioModeler 3.1 - Microsoft SQL Server Driver.
/* Document   : C:\Data\Personal\Education\Thesis-II\WIP\Graphics\Fina
/* Time Created: July 27, 2003 10:03 PM.
/* User Action : From VisioModeler Generate wizard.
/* Connected Data source : grocery
/* Connected Server      : RPAYNER
/* Connected Database    : grocery

/* Create grocery database.
use master
go
create database grocery
go
```

Figure 5-12: DDL Script Generated by VisioModeler

## 5.6. STEP 4: Building an OLAP Cube

Having created relational tables to house our multidimensional data in SQL Server, we now complete the last step of our multidimensional lifecycle. In this step we build an OLAP cube from the physical schema to store our data in Decision Support System (DSS) format. Our OLAP cube will allow us to analyze the data as originally described using the modeling constructs of our MORM conceptual model.

The remainder of this section guides us through the process of creating and using the cube to analyze data from our Grocery example. We briefly outline operations

necessary for setting up data connections, designing cube structure, processing cubes and finally analyzing cube data with SQL Server Analysis Manager.

### 5.6.1. Setting up the Database & Data Source

Before building the cube, our initial steps include setting connections to the source of our data in ODBC Data Source Administrator. Using Analysis Manager, an Analysis Services program that manages OLAP objects and data, we then create a new Retail Grocery database object (shown in Figure 5-13) to hold data sources, cubes, and shared dimensions together.

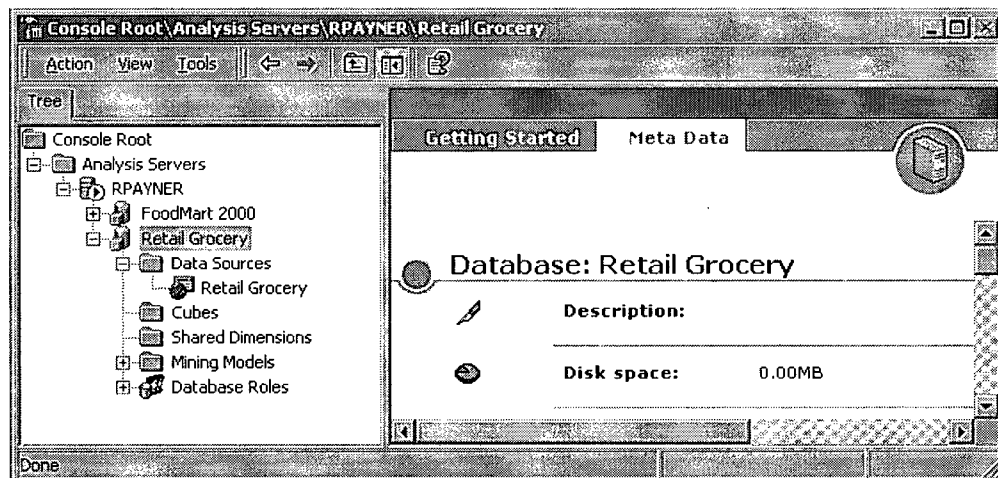


Figure 5-13: Analysis Services Database Object

With the database object defined we establish a data source in Analysis Manager that connects our Grocery database to the system data source name previously created. This allows us to access all data from this source as we build our cube.

### 5.6.2. Building the Cube

Analysis Manager's Cube Wizard is used to build our cube by defining its measures and dimensions. We first define the source of our measures through the Wizard by selecting our *Sales Transaction* event table from our data source. Measures are then defined for our cube by selecting the *price*, *cost*, *profit*, and *quantity* numeric columns.

We begin building dimensions by first creating the *Time* dimension. To do this we create a new dimension in the wizard and define hierarchy levels by selecting *year*, *quarter*, and *month*. We then designate this dimension as shared so we may access it in other cubes in our implementation. *Product*, *Customer*, and *Store* dimensions are then created in a similar fashion. Upon creating event measures and related dimensions, we confirm the design of the cube through the Cube Editor (shown in Figure 5-14), which contains our POS Retail Sales cube structure.

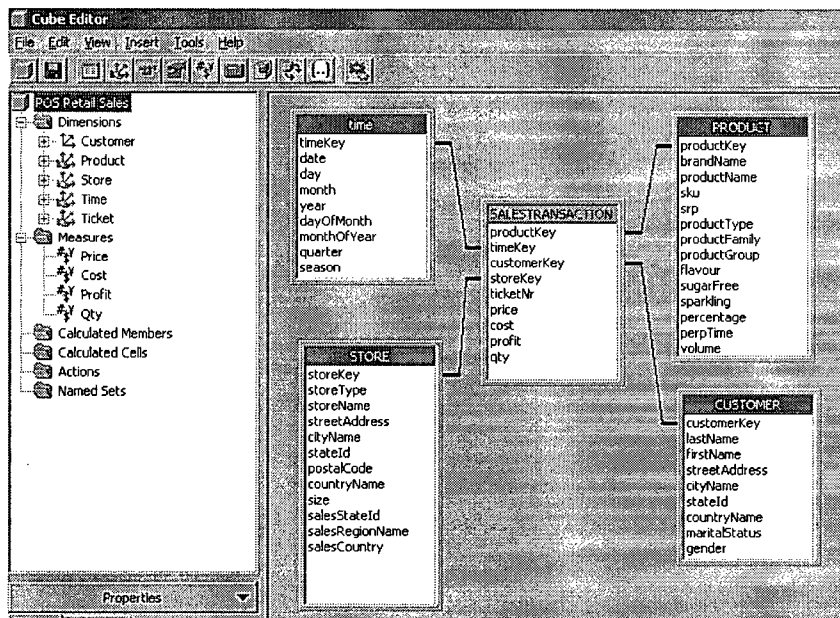


Figure 5-14: Analysis Services Cube Editor

### 5.6.3. Designing Storage and Processing the Cube

With the structure of our Retail Sales cube designed, our next steps are to design storage options for the data and aggregations of our cube, then populate it with data. Shown in Figure 5-15, we use the Design Storage Wizard to designate MOLAP for our storage mode, create the aggregation design for the Sales cube, and then process the cube.

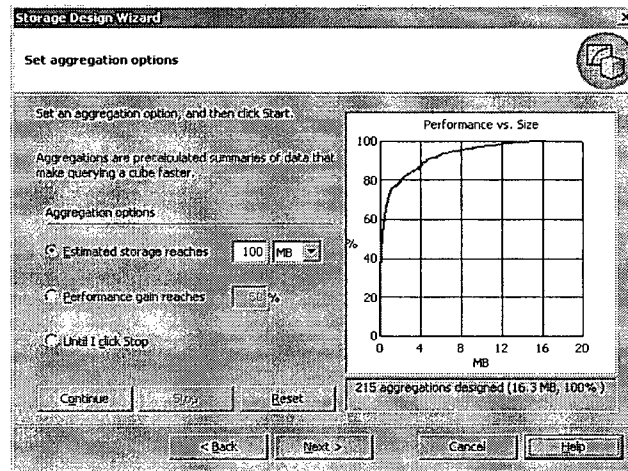


Figure 5-15: Cube Processing Using the Design Wizard

Processing the cube loads our Grocery data and calculates summary values. These pre-calculated summaries of data will greatly improve the efficiency and response time of queries. The results of our cube processing are shown in Figure 5-16.

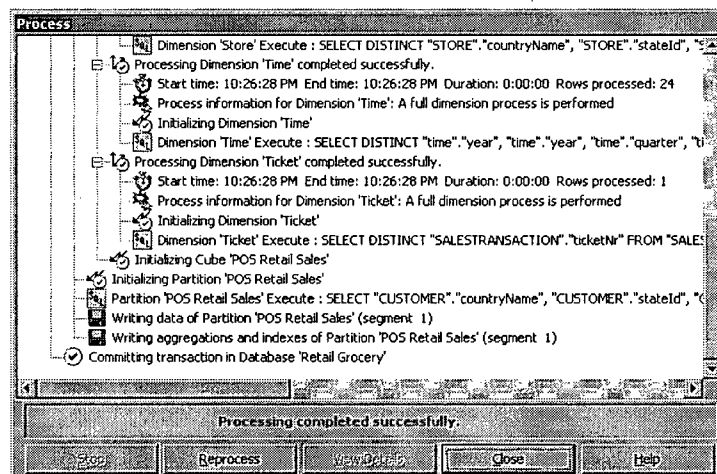


Figure 5-16: Final Cube Processing Results

### 5.6.4. Browsing Cube Data

With our cube processed, we can analyze data in many different ways. Using the Cube Browser we can perform various OLAP operations, including filtering the amount of dimension data, drilling down to see greater detail, and drilling up to see less. Figure 5-17 illustrates filtering by Time where data is filtered to for a particular quarter. Figure 5-18 depicts drill-down in which we expand the Drink group to include its families.

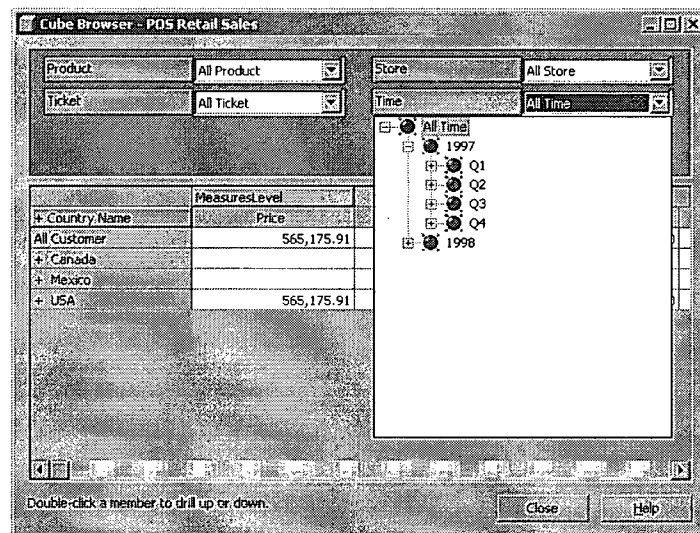


Figure 5-17: Filtering Example Within the Cube Browser

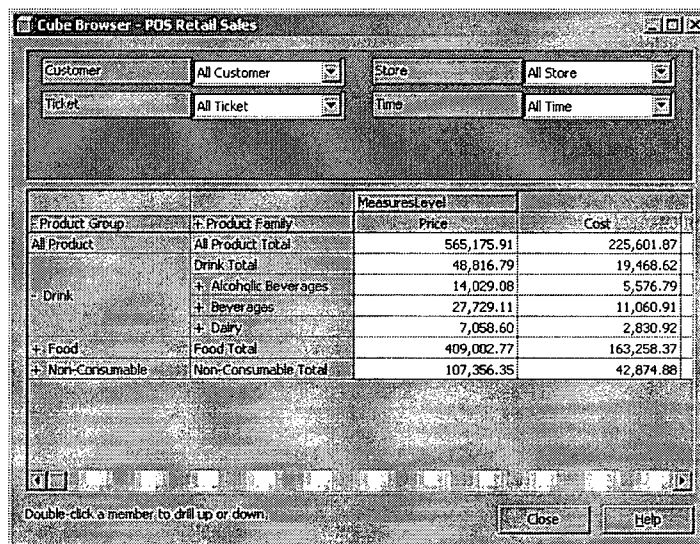


Figure 5-18: Drill Down Example Within the Cube Browser

## **5.7. An Evaluation of Our Case Study**

Through a case study implementation, we have illustrated how our conceptual fact-oriented approach simplifies conceptual design when modeling data warehouses. The implementation of our model allowed us to put into practice the ideas proposed by our conceptual modeling approach. Appendix C summarizes the implementation results from our case study. Included in the Appendix are details per multidimensional requirement for the four stages of our implementation - conceptual, logical, physical, and OLAP.

While we were able to implement our three newly introduced MORM constructs with relative ease, we conclude the generation process from a conceptual model to an OLAP tool is not immediate for all multidimensional concepts. This is mainly because certain multidimensional constructs in our conceptual model are implemented differently, or not at all, in our OLAP tool. As Hahn, Sapia, and Blaschka have found (2000), there are several mismatches between the data models of commercial OLAP tools and conceptual graphical modeling notations. Specifically, tools do not often provide sufficient native constructs to represent each element of a graphical notation. This implies that the generation process must perform a mapping between the semantics of the graphical notation and the tool configuration, most often with a loss of semantics.

Commercial OLAP products provide their own methods of assessing multidimensional semantics and concepts. In addition to database structures, OLAP tools implement underlying metadata that provide key multidimensional semantics (e.g. measures and dimensions). For proprietary reasons, each tool may implement these semantics and properties differently. Ideally, proper multidimensional design uses a conceptual approach totally independent of implementation concerns, allowing the direct generation into commercial OLAP tools.

Several expressiveness differences between our conceptual model and our Analysis Services OLAP tool proved to be our greatest implementation challenge. Specifically, several concepts used in our conceptual design lacked a corresponding Analysis Services representation. In most cases, however, we managed to find transformations that preserve a large part of the original model semantics and our results show that a generation process is generally feasible and useful. We note the following three areas where our OLAP tool did not have a corresponding multidimensional representation and a transformation was required during our implementation.

#### ***5.7.1. Hierarchies: Multiple, Alternative Path, and Shared***

Multiple, alternative path, and shared hierarchies are not directly supported in Analysis Services but we are able to indirectly implement them and address our multidimensional requirements while preserving as much of the original semantics as possible.

For all three hierarchy types, the logical and physical models implement them as table columns and the hierarchy levels are implicit in the flat table design. In the OLAP tool, multiple path hierarchies are defined as two or more dimensions with names that share the same dimension prefix but have different suffixes (e.g. Time.Calendar and Time.Season). Two hierarchies must also be defined for alternative path hierarchies since our tool can only handle dimensions with a tree structure (i.e. different hierarchies cannot merge in an endpoint). In this case we have to duplicate the dimension beginning at the merging point. As for shared hierarchies, two or more separate dimensions are also implemented starting at the dimension level where the merging occurs. The underlying dimension tables as well as any aggregations are shared in all cases.

### 5.7.2. Non-Strictness

Non-strictness is not supported in our case study. While we have proposed an approach to model this property at the conceptual level using a many-to-many uniqueness constraint (see section 4.4.3.2), our chosen OLAP tool does not provide adequate aggregation support for such hierarchies. Considering the lack of OLAP support and our adherence to a simple star schema design, we require all hierarchies to be strict in our implementation. As such, non-strict hierarchies must be converted to strict hierarchies if aggregations are applicable. Given the lack of support for non-strictness in the OLAP market, we provide additional background supporting this aspect of our case study.

In a non-strict hierarchy there are many-to-many relationships between the different levels in a dimension where a lower-level item can be a member of several items at a higher-level (e.g. a sales region may cross several states and a state may be in several sales regions). Traditionally, OLAP tools only permit strict hierarchies where every lower-level item belongs to a single higher-level item. Such is the case with Analysis Services, which explicitly requires strict hierarchies and does not address the issue of correct aggregation for non-strict hierarchies.

As such, we only permit strict hierarchies in our implementation. Our underlying assumption is that adherence to the simple star schema design requires dimension hierarchies to be strict. Our argument is consistent with that of Lenz and Shoshani (1997) who argue that the premise underlying the applicability of aggregation is *summarizability*, which essentially means lower-level results can be directly combined into higher-level results. For this to be true, one lower-level dimension value must map to exactly one higher-level value. Having irregular, many-to-many dimension levels violates this characteristic of OLAP data.



In reviewing the literature, only one technique was found that addresses the issue of summarizability in non-strict hierarchies. Pedersen, Jensen, and Dyreson (1999) present a technique and associated algorithm for achieving summarizability by adding dummy values and “fusing” sets of parents together. The basic idea of this technique is to combine a set of parent values into one “fused” value, link the child value to this new value, then insert the fused values into a new category between the child and parent categories. These transformations require major restructuring of the hierarchy and violate our pure star schema design principle. The complexity of this technique defeats the benefits of our approach, possibly leading to incorrect results during aggregation through double counting. If summarizability is relevant, users should be able to analyze the data and obtain correct results without having to worry about such double counting.

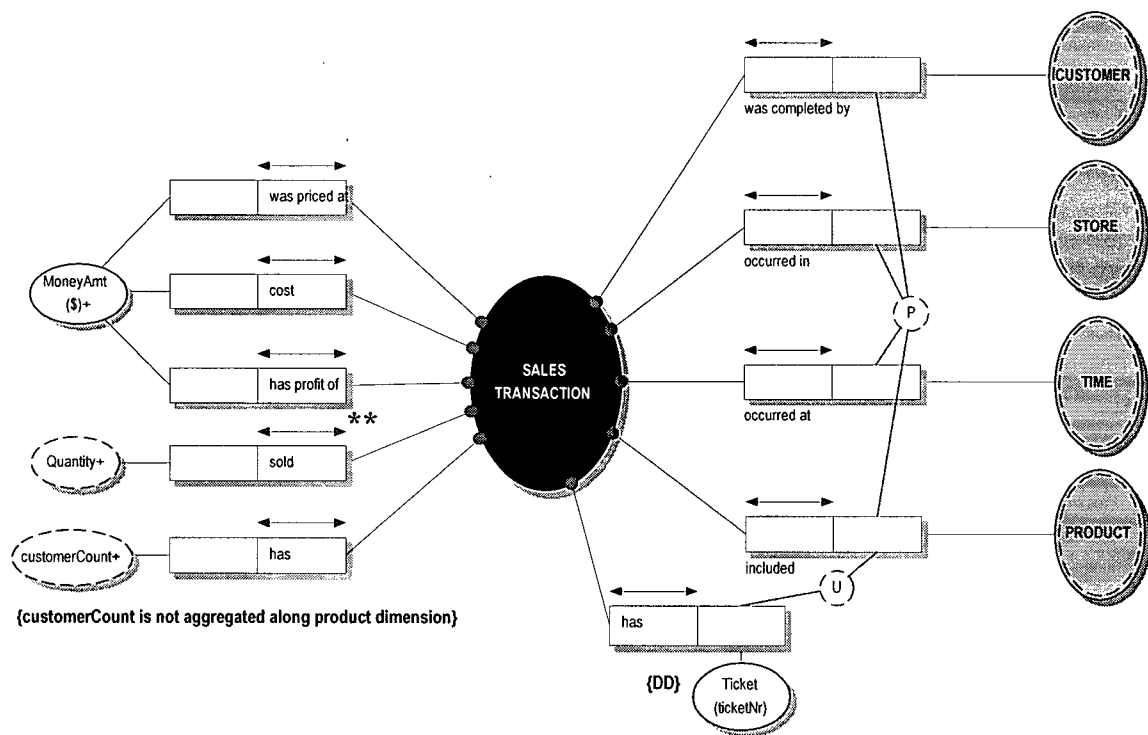
### ***5.7.3. Many-to-Many Relationships Between Events and Dimensions***

We give this topic considerable attention since having many-to-many relationships between a dimension and an event causes several difficult issues during multidimensional implementation. These issues include losing the standard star schema structure, increasing the complexity of query formation and degrading query performance by adding joins (Song, Rowan, Medsker, & Ewen, 2001). Therefore, it is desirable to handle these many-to-many relationships while keeping the structure of the star schema.

Song et al. (2001) investigate several methods of handling many-to-many relationships and discuss the relative advantages and disadvantages of each. A key argument of theirs is that to maintain the star schema structure, relationships between events and dimensions should be made many-to-one and events should be mapped to the

lowest categories in the dimensions. We follow this solution approach in which we lower the grain of the Sales event to the lowest dimension grain level (i.e. product).

To illustrate, while it is possible for our Sales event to be at the ticket grain (i.e. have multiple products per sale), we lower the grain of the event to the line item level so there are multiple records (i.e. multiple line items) relating to that specific event. This ensures we now have a many-to-one relationship between Sale and Product. Figure 5-19 illustrates how we have achieved this at the conceptual level.



**Figure 5-19: Modeling Many-to-Many Relationships**

The external uniqueness constraint (circled "p") on the Store, Customer, Product, and Time dimensions indicates each transaction occurs for *at most one* Store, Customer, Product, Time. Thus, the combination of Product, Store, Customer and Time is unique for each Sales Transaction. The "p" indicates this combination is the primary indicator for each event. As indicated by the additional uniqueness constraint (circled "u") on

Ticket and Product, the combination of ticket number and Product is also unique. This means one ticket can relate to more than one Product and indicates the many-to-many relationship between ticket and product.

We also include the constraint {DD} to identify ticket as a degenerate dimension of the event. We do this to ultimately generate an OLAP ticket dimension so we can group line items and determine the ticket total. In the OLAP tool the ticket becomes a dimension with only one hierarchy level, allowing us to group multiple measures per ticket. Using this approach the conventional star schema is retained, providing a clear logical view of the business process and allowing implementation in our OLAP tool.

## **5.8. Summary**

In this chapter we tested the practicality and usability of our work by applying our conceptual multidimensional modeling approach to a case study to solve a data analysis problem. Using our MORM guidelines, we developed a conceptual model and mapped it to a logical schema in VisioModeler. From the VisioModeler models we generated a physical star schema in Microsoft SQL Server 2000 and subsequently built an OLAP cube in SQL Server 2000 Analysis Services. Largely due to maturity, functionality, and availability, we chose VisioModeler for conceptual and logical modeling, while Microsoft SQL Server and Analysis Services were our choices for physical and OLAP implementation. In spite of some cube generation limitations with our chosen OLAP tool, the implementation demonstrated that our approach naturally and expressively models the main structural properties of multidimensional data at the conceptual level and serves as the basis for subsequent design phases.

## **6. CONCLUSIONS & FUTURE RESEARCH**

### **6.1. Thesis Summary**

The primary focus of this thesis has been the development of a fact-oriented approach to modeling the structural properties of multidimensional data at the conceptual level. Our main objective was to provide a natural, simple, and expressive modeling approach to address the fundamental deficiencies of existing multidimensional models. We have accomplished our objective through an exploration of multidimensional concepts and the development of a modeling approach that simplifies multidimensional design by using natural language, intuitive diagrams and example data populations.

To better understand the functionality of data warehouses and OLAP applications we have provided an overview of their logical and physical architectures and the main processes associated with their use. Our overview described data source, data storage, application, and presentation layers and discussed how physical OLAP architectures map onto these layers in several ways. We also discussed data staging services that get data into the data warehouse and query services which focus on getting data out. Our overview highlighted the differences between data warehouses and traditional OLTP applications and, due to the significant differences in underlying data structures, we concluded different conceptual modeling techniques are required for data warehouses.

Our attention then turned to understanding data modeling techniques and we examined basic data modeling concepts by looking at conceptual, logical, and physical information levels. We emphasized the importance of data modeling at the conceptual level and provided an overview of several conventional data modeling approaches with a specific focus on ER, UML and ORM. To better understand multidimensional data and

its semantic differences we presented the properties of multidimensional data through an example. Using the analysis requirements demonstrated with a sample Grocery chain we revealed a set of multidimensional concepts that included events, dimensions, measures, additivity, derived measures, classification hierarchies, strictness, completeness and the categorization of dimensions. To understand how existing models address multidimensional concepts we reviewed the current state of multidimensional modeling literature. We briefly reviewed logical, physical, and formal works, but our main focus was on models attempting to express semantics at the conceptual level. Based on our review we concluded that a natural and complete conceptual design technique does not exist that adequately conceptualizes and clearly communicates multidimensional designs to both business and technical users. In addition, existing works presented few design guidelines to ensure their approaches are properly and easily applied.

The fundamental deficiencies and shortcomings of existing techniques in formulating, transforming and evolving a conceptual model provides motivation for our work. Inspired by ORM, we introduced our fact-oriented MORM approach as a specialization of ORM by defining additional graphical constructs and guidelines to consider the unique characteristics of multidimensional data. To support the semantics inherent in multidimensional data we introduced three MORM constructs - the Event Object Type, the Dimension Entity Type, and the Hierarchy Object Type. These constructs represent the events, dimensions, and classification hierarchies we are interested in analyzing. Using our Grocery example, we demonstrated how our approach models each of the multidimensional requirements previously revealed. We have supplemented our MORM model with several key design guidelines to guide data

modelers in using our method to develop multidimensional models. Our guidelines provide various levels of abstraction and simplify conceptual design by distinguishing five design levels; preliminary segmentation, business process family definition, business process definition, event definition, and dimension definition.

We have tested the practicality and usability of our approach by applying it to a case study to solve a data analysis problem. Using our Grocery example, we have demonstrated that our approach can be easily implemented using existing technologies. We chose VisioModeler for conceptual and logical modeling, while Microsoft SQL Server and its OLAP component Analysis Services were our choices for physical and OLAP cube implementation. Using our MORM guidelines, we have developed a conceptual model and mapped it to a logical schema in VisioModeler. From the mapped logical model we generated a physical star schema in Microsoft SQL Server 2000, and subsequently built an OLAP cube in Analysis Services that allowed us to analyze Grocery data as described in our original MORM model. In spite of some cube generation limitations with our chosen OLAP tool, the implementation demonstrated the practicality of our approach as the basis for subsequent data design phases.

## **6.2. Contributions**

To the best of our knowledge, we have presented the first fact-oriented approach to conceptual multidimensional modeling. We believe leveraging the fact-oriented paradigm provides us with a conceptual multidimensional model that is more natural and expressive than existing multidimensional models. As such, MORM provides a solid basis for solving conceptual multidimensional modeling problems with a more natural and expressive conceptual model than existing approaches. Examining multidimensional

data in terms of elementary facts provides a truly conceptual approach and simplifies the analysis and design process by using natural language, intuitive diagrams, and real-world data examples. We believe our fact-oriented approach will help designers capture and satisfy complex modeling requirements, help business users better understand the structure and navigation paths of the data warehouse, and facilitate communication between business users and data modelers.

Another major contribution of our work stems from our use of a widely accepted modeling technique. By specializing ORM, we minimize the effort required of data modelers to learn a new modeling notation for multidimensional data. Our approach requires a shallow learning curve since data modelers can combine MORM elements with classical ORM elements and, although the approaches will be different, data models for OLTP and OLAP applications can be specified using a uniform notation.

Another of our contributions is the provision of design guidelines to construct multidimensional models using our approach. We believe these guidelines reflect the natural way users and data modelers think about multidimensional data and lead to a simple yet powerful multidimensional model. Whereas other approaches use flat design, our guidelines produce multilevel subschemas that group different levels of abstraction and ultimately simplify the conceptual design of large data warehouses.

Finally, we have successfully demonstrated that our approach can be implemented using existing data modeling tools and database technologies. Through a case study, we have developed a conceptual model and mapped it to a logical schema in VisioModeler, a well-known data modeling tool. We have generated a physical star schema in Microsoft SQL Server 2000 and subsequently built an OLAP cube in SQL Server 2000 Analysis

Services. By putting all ideas developed throughout this thesis into practice, we have proven that our approach suggests a new way of modeling multidimensional data.

### **6.3. Limitations and Future Research**

While we were able to easily implement conceptual, logical and physical schemas using our approach, the OLAP cube generation process was not immediate and further work in this area is encouraged. In future it would be beneficial to investigate a generation process that automatically transforms semantics at the conceptual into a generic OLAP model compatible with the majority of commercial OLAP tools. The challenges in this area are due to the fact that complex multidimensional constructs in conceptual models are not supported or are implemented inconsistently in OLAP tools.

While our approach was successfully tested using a case study and several real world implementations, the limited number of examples limits our work. To further examine the practicality of our approach and demonstrate its benefits, other case studies should be carried out using data from different industries. As part of this investigation, it would be particularly useful to examine complex data in which there are no natural numeric measurements associated with events and non-numeric measures must be used.

It would also be beneficial to investigate an extension of our model to represent the dynamic properties of data warehouses and OLAP applications. These dynamic aspects could include the definition of initial user requirements and subsequent OLAP operations (e.g. roll-up, drill-down, slice-dice, pivoting) for further analyzing data.

While we have provided high-level design guidelines, future work can also build on these guidelines to develop a complete multidimensional design methodology. Based on the MORM model introduced, a methodology could include a complete process that



explicitly considers all the underlying design guidelines hidden in our approach. As part of this methodology, specific rules could be developed for first identifying business process families and business processes, then subsequently deriving dimensions and events from them.

## BIBLIOGRAPHY

- Abello, A., Samos, J., & Saltor, F. (2001). A framework for the classification and description of multidimensional data models. *Proceedings of the 12<sup>th</sup> International Conference on Database and Expert Systems Applications (DEXA)*, 668-677.
- Agrawal, A., Gupta, A., & Sarawagi, S. (1997). Modeling multidimensional databases. *Proceedings of the 13<sup>th</sup> International Conference on Data Engineering (ICDE)*, 232-243.
- Barker, R. (1990). *CASE\*Method: Tasks and deliverables*. Wokingham, England: Addison Wesley.
- Batani, C., Ceri, S., & Navathe, S. (1992). *Conceptual database design: An entity relationship approach*. Redwood City, CA: Benjamin Cummings.
- Batra, D., Hoffer, J. & Bostrom, R. (1990). Comparing representations with relational and EER models. *Communications of the ACM*, 33(2), 126-139.
- Becker, Scot A. (2000). Arguments against the use of ORM (and their rebuttals). *Journal of Conceptual Modeling*. <http://www.inconcept.com/JCM/June2000/becker.html>
- Bernus, P., Mertins, K., & Schmidt, G. (Eds.). (1998). *Handbook on architectures of information systems*. Berlin: Springer-Verlag
- Blaschka, M., Sapia, C., Hofling, G., & Dinter, B. (1998). Finding your way through multidimensional data models. *Proceedings of the 9<sup>th</sup> International Conference on Database and Expert Systems Applications (DEXA '98)*, 198-203.
- Boehnlein, M., & Ulbriche-vom Ende, A. (1999). Deriving initial data warehouse structures from the conceptual data models of the underlying operational information systems. *Proceedings of the 2<sup>nd</sup> International Workshop on Data Warehousing and OLAP (DOLAP'99)*, 15-21.
- Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified modeling language user guide*. Reading, MA: Addison-Wesley.
- Bulos, D. (1996). OLAP database design: A new dimension. *Database Programming and Design*, 9(6), 32-37.
- Cabibbo, L., & Torlone, R. (1999). A framework for the investigation of aggregate functions in database queries. *Proceedings of the 7<sup>th</sup> International Conference on Database Theory (ICDT-99)*, 383-397.

- Chan, C., & Ioannidis, Y. (1998). Bitmap index design and evaluation. *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD '98)*, 355-366.
- Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology." *ACM SIGMOD Record*, 26(1), 65-74.
- Chen, P. (1976). The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1), 9-36.
- Codd, E., (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377-387.
- Data Warehousing Institute. (2000). *Data warehousing: what works? (9)*. The Data Warehousing Institute.
- Dyreson, C. (1996). Information retrieval from an incomplete data cube. *Proceedings of the 22<sup>nd</sup> International Conference On Very Large Databases (VLDB'96)*, 532-543.
- Elmasri, R., & Navathe, S. (1994). *Fundamentals of database systems* (2nd ed.). Menlo Park, CA: Benjamin Cummings.
- Finkelstein, C. (1989). *Introduction to information engineering*. Reading, MA: Wesley
- Gingras, F., & Lakshmanan, L. (1998). nD-SQL: A multi-dimensional language for interoperability and OLAP. *Proceedings of the 24<sup>th</sup> International Conference On Very Large Databases (VLDB'98)*, 134-145.
- Golfarelli, M., Maio, D., & Rizzi, S. (1998a). Conceptual design of data warehouses from E/R schemes. *Proceedings of the 31<sup>st</sup> Hawaii International Conference on System Sciences*, 334-343.
- Golfarelli, M., Maio, D., & Rizzi, S. (1998b). The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 7(2-3), 215-247.
- Hahn, K., Sapia, C., & Blaschka, M. (2000). Automatically generating OLAP schemata from conceptual graphical models. *Proceedings of the 3<sup>rd</sup> ACM International Workshop on Data Warehousing and OLAP*, 9-16.
- Halpin, T. (1995). *Conceptual schema and relational database design* (2nd ed.). Sydney: Prentice Hall.
- Halpin, T., & Bloesch, A. (1999). Data modeling in UML and ORM: A comparison. *Journal of Database Management*, 10(4), 4-13.

- Halpin, T. (2001). *Information modeling and relational databases: From conceptual analysis to logical design*. San Francisco: Morgan Kaufmann.
- Hay, D. (1999). Object orientation and information engineering: UML. *The Data Administration Newsletter*, (9). <http://www.tdan.com>
- Inmon, W. (1996). *Building the data warehouse*. New York: John Wiley & Sons.
- Kimball, R. (1996). *The data warehouse toolkit: Practical techniques for building dimensional data warehouses*. New York: John Wiley & Sons
- Kimball, R. (1997). A dimensional modeling manifesto. *DBMS and Internet Systems*, <http://www.dbmsmag.com>.
- Kimball, R., Reeves, L., Ross, M., & Thornthwaite, W. (1998). *The data warehouse lifecycle toolkit*. New York: John Wiley & Sons
- Kimball, R., & Ross, M. (2002). *The data warehouse toolkit: The complete guide to dimensional modeling* (2nd ed.). New York: John Wiley & Sons
- Lenz, H., & Shoshani, A., (1997). Summarizability in OLAP and statistical databases. *Proceedings of the 9th International Conference on Scientific and Statistical Databases*, 39–48.
- Microstrategy, Inc. (1995). The case for relational OLAP. <http://www.strategy.com>
- Martin, J. (1990). *Information engineering*. Englewood Cliffs: Prentice Hall.
- NIST. (1993). *Integration definition for information modeling (IDEFIX)*. FIPS Publication 184. National Institute of Standards and Technology.
- Pedersen, T., & Jensen, C. (1999). Multidimensional data modeling for complex data. *Proceedings of the 15<sup>th</sup> IEEE International Conference on Data Engineering (ICDE'99)*, 336–345.
- Pedersen, T., Jensen, C., & Dyreson, C. (1999). Extending practical pre-aggregation for on-line analytical processing. *Proceedings of the 25<sup>th</sup> International Conference on Very Large Databases (VLDB'99)*, 663–674.
- Raden, N. (1995). Modeling a data warehouse. <http://www.archerdecision.com/artic3.htm>.
- Ritson, P., & Halpin, T. (1993). Mapping integrity constraints to a relational schema. *Proceedings of the 4<sup>th</sup> Australian Conference on Information Systems (ACIS'93)*, 381–400.
- Sapia, C., Blaschka, M., Höfling, G., & Dinter, B. (1998). Extending the ER model for the multidimensional paradigm, *Proceedings of the 1<sup>st</sup> International Workshop on Data Warehouse and Data Mining (DWD'98)*, 105–116.

- Song, I., Rowan, W., Medsker, C., & Ewen, E. (2001). An analysis of many-to-many relationships between fact and dimension tables in dimensional modeling. *Proceedings of the 3<sup>rd</sup> International Workshop on Design and Management of Data Warehouses (DMDW'01)*, 6.1-6.13.
- Teorey, T., Yang, D., & Fry, J. (1986). A logical design methodology for relational databases using the extended entity-relationship model. *Computing Surveys*, 18(2), 197-222.
- Theodoratos, D., & Sellis, T. (1999). Dynamic data warehouse design. *Proceedings of the 1<sup>st</sup> International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99)*, 1-10.
- Trujillo, J., Palomar, M., & Gómez, J. (2000). Applying object-oriented conceptual modeling techniques to the design of multidimensional databases and OLAP applications. *Proceedings of the 1st International Conference on Web-Age Information Management (WAIM 00)*, 83-94.
- Tryfona, N., Busborg, F., & Christiansen, J. (1999). starER: A conceptual model for data warehouse design. *Proceedings of the 2<sup>nd</sup> International Workshop on Data Warehousing and OLAP (DOLAP'99)*, 3-8.
- Widom, J. (1995). Research problems in data warehousing. *Proceedings of the 4<sup>th</sup> International Conference in Information and Knowledge Management (CIKM'95)*, 25-30.
- Wu, M., & Buchmann, A. (1997). Research issues in data warehousing. *Proceedings of the 7<sup>th</sup> German Conference on Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'97)*, 61-82.

## APPENDIX A: ORM CONSTRUCTS

This appendix summarizes ORM's main constructs as described in chapter 4 of Bernus, Mertins, & Schmidt (1998). Shown in Figure A-1, constructs are labeled with a number and further described in Table A-1.

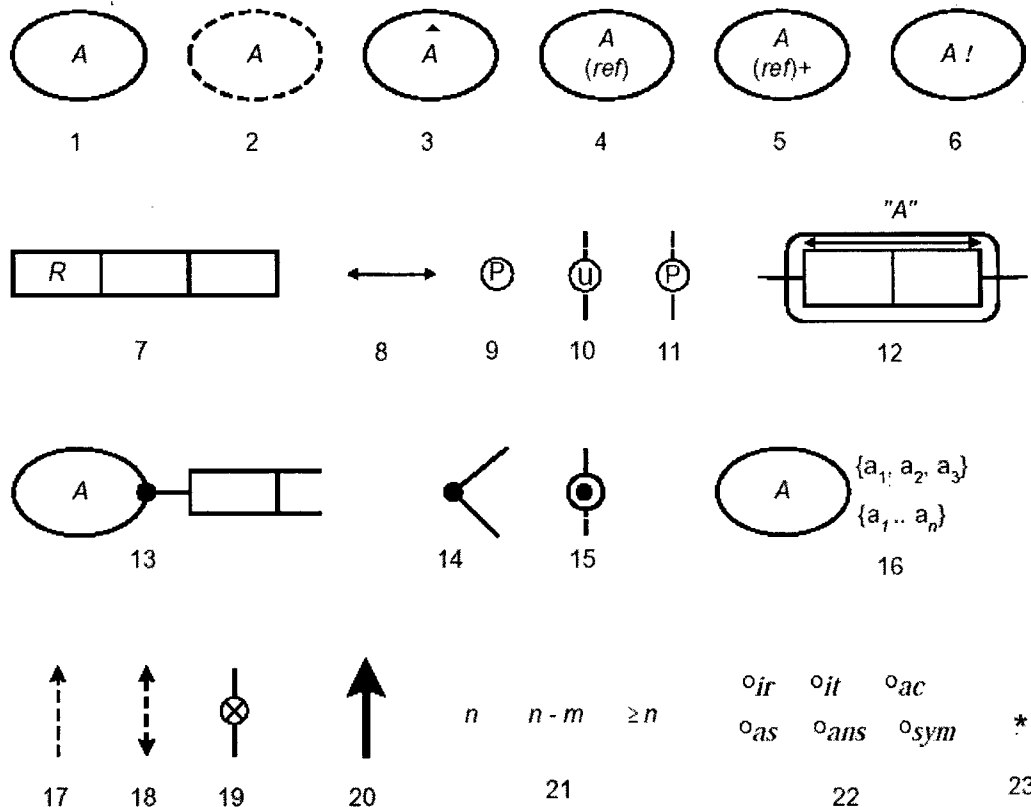


Figure A-1: Graphical Notation of ORM Constructs

#	Construct	Description
1	Entity Type	Tangible or abstract object that is identified by a definite description (e.g. the student with studentID 25899). Descriptions typically indicate the entity (e.g. student), a value (e.g. 25899) and a reference mode (e.g. studentID).
2	Value Type	Denotes a lexical object type (e.g. a character string or number) that is identified by <i>constants</i> (e.g. David R. Williams, 25899). Another notation for value types encloses the value type name in parentheses.
3	Duplicate Object Type	Object types that appear more than once in the schema may be tagged with an arrow tip that "points" to the existence of another occurrence.
4	Reference Mode	Each entity type must have at least one <i>reference scheme</i> that indicates how each instance of the entity type may be mapped via predicates to a

#	Construct	Description
		combination of one or more values. Reference schemes are abbreviated by displaying the <i>reference mode</i> in parentheses beside the name of the entity type. The reference mode indicates how values relate to the entities.
5	<i>Numeric Value</i>	A plus sign "+" may be added if values are numeric
6	<i>Independent Entity Type</i>	Means instances of that type may exist without participating in any facts.
7	<i>Predicate</i>	Depicts a ternary <i>predicate</i> comprised of three <i>roles</i> . Each role is depicted as a box, and must be played by exactly one object type. Roles are connected to their players by a line segment.
8	<i>Internal Uniqueness Constraints</i>	Arrow tipped bars placed over one or more roles in a predicate declare that instances for that role (combination) in the relationship type population must be unique.
9	<i>Primary Uniqueness Constraints</i>	A predicate may have one or more uniqueness constraints, at most one of which may be declared <i>primary</i> by adding a "P".
10	<i>External Uniqueness Constraint</i>	A circled "u" may be applied to two or more roles from different predicates by connecting to them with dotted lines. Instances of the combination of those roles in the join of those predicates are unique.
11	<i>Primary External Uniqueness Constraint</i>	To declare an external uniqueness constraint primary, use "P" instead of "u".
12	<i>Objectified Predicates</i>	If we wish to talk about a relationship type we may <i>objectify</i> it (i.e. make an object out of it) so that it can play roles. Graphically, the objectified predicate is enclosed in a rounded rectangle.
13	<i>Mandatory Role Constraint</i>	Declares that every instance in the population of the role's object type must play that role.
14	<i>Disjunctive Mandatory Constraint</i>	Applied to two or more roles to indicate that all instances of the object type population must play <i>at least one</i> of those roles. This may often be shown by connecting the roles to a black dot on the object type
15	<i>Disjunctive Mandatory Constraint</i>	Another way to indicate all instances of the object type population must play <i>at least one</i> of those roles, here by connecting the roles by dotted lines to a circled black dot.
16	<i>Value Constraints</i>	To restrict an object type's population to a given list, the relevant values may be listed in braces. If the values are ordered, a range may be declared separating the first and last values by ".."
17	<i>Subset Constraint</i>	A dotted arrow from one role sequence to another is a <i>subset constraint</i> , restricting the population of the first sequence to be a subset of the second.
18	<i>Equality Constraint</i>	A double-tipped arrow is an <i>equality constraint</i> , indicating the populations must be equal.
19	<i>Exclusion Constraint</i>	A circled "X" is an <i>exclusion constraint</i> , indicating the populations are mutually exclusive. Exclusion constraints may be applied between two or more sequences.
20	<i>Subtype</i>	A solid arrow from one object type to another indicates that the first object type is a (proper) <i>subtype</i> of the other.
21	<i>Frequency Constraint</i>	Applied to a sequence of one or more roles, these indicate that instances that play those roles must do so exactly <i>n</i> times, between <i>n</i> and <i>m</i> times, or at least <i>n</i> times.
22	<i>Ring Constraint</i>	May be applied to a pair of roles played by the same host type. These indicate that the binary relation formed by the role population must be irreflexive (ir), intransitive (it), acyclic (ac), asymmetric (as), antisymmetric (ans) or symmetric (sym).
23	<i>Derivable Fact Type</i>	An asterisk "*", placed beside a fact type indicates it is derivable from other fact types.

**Table A-1: ORM Constructs and Associated Descriptions**

# **APPENDIX B: MORM SCHEMA FOR RETAIL CASE STUDY**

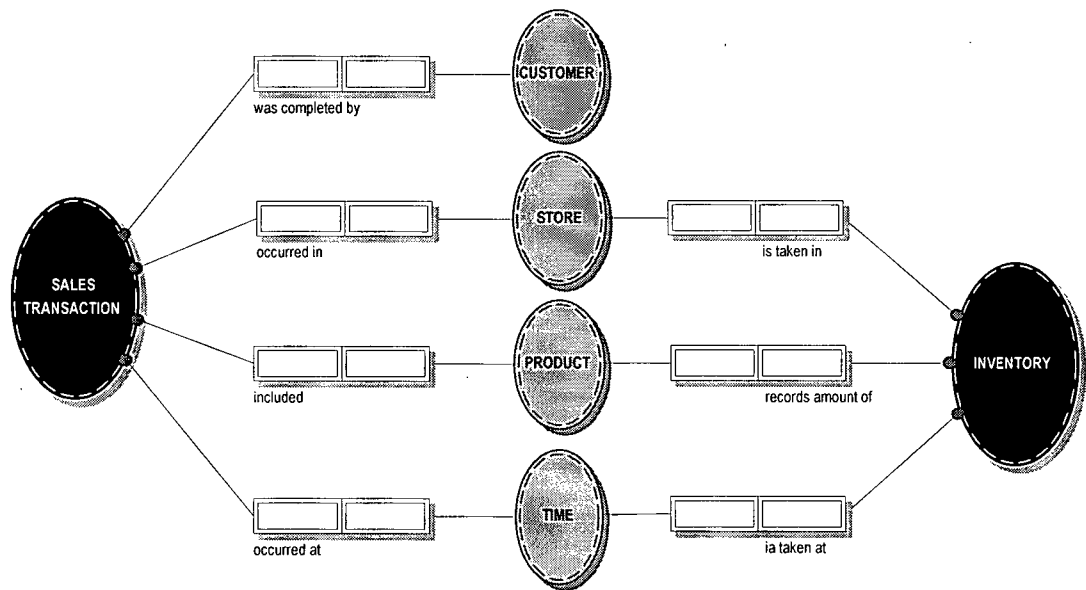


Figure B-1: MORM Level 1 - Retail Business Process Family

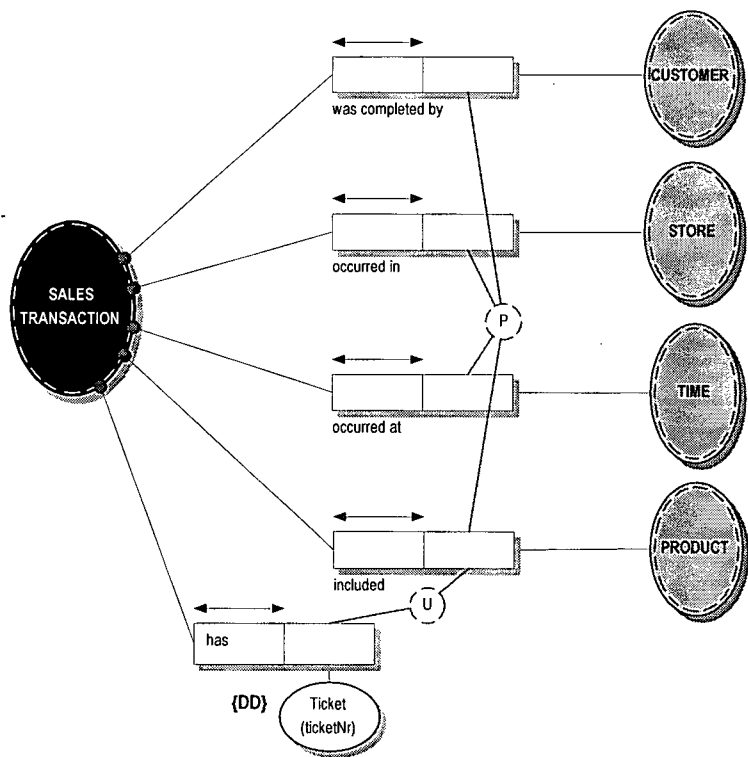


Figure B-2: MORM Level 2 - POS Retail Sales Business Process



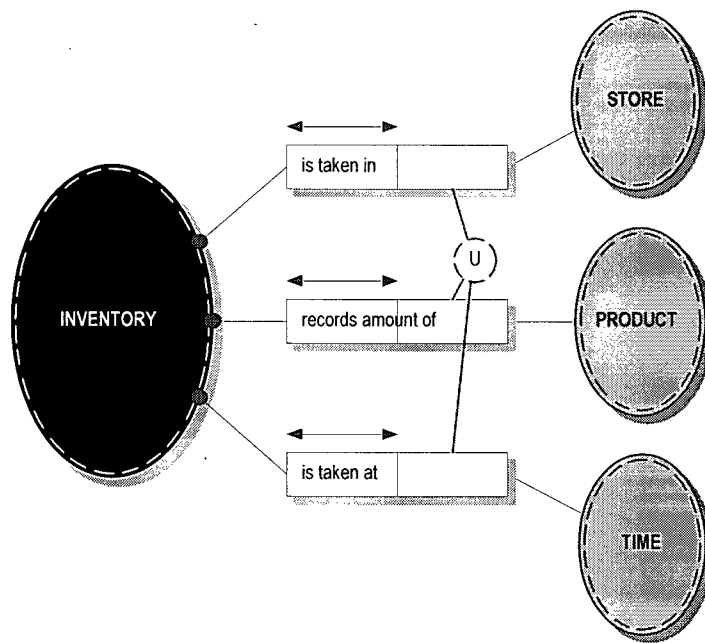
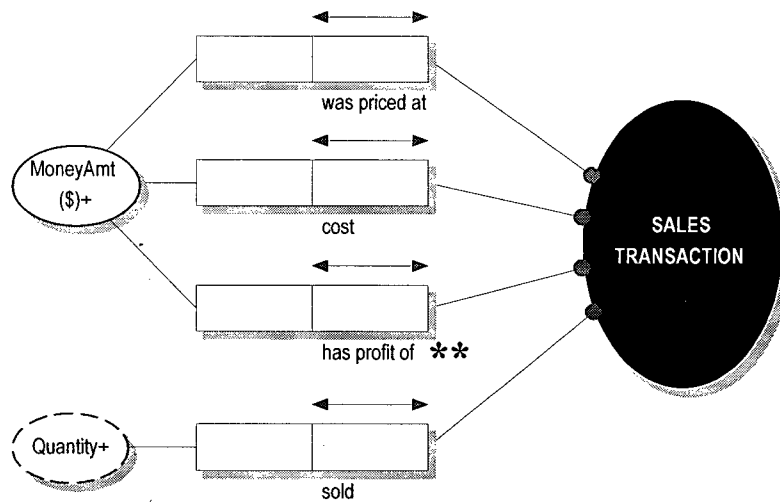


Figure B-3: MORM Level 2 - Inventory Business Process



\*\* { profit = price - cost }

define Sales Transaction has profit of MoneyAmt as  
 Sales Transaction was priced at MoneyAmt1, an  
 Sales Transaction cost MoneyAmt2, and  
 MoneyAmt = MoneyAmt1 - MoneyAmt2

Figure B-4: MORM Level 3 - Sales Transaction Event

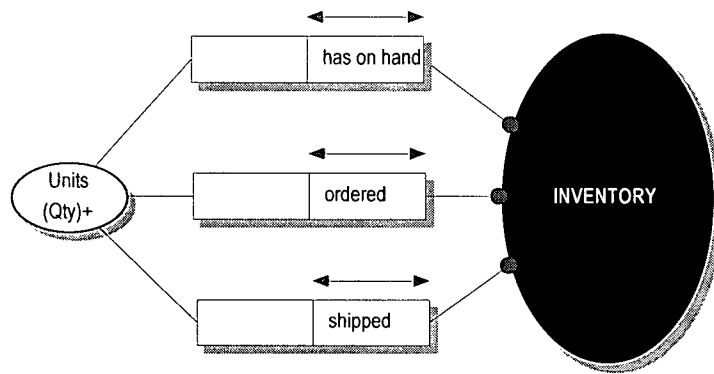


Figure B-5: MORM Level 3 - Inventory Event

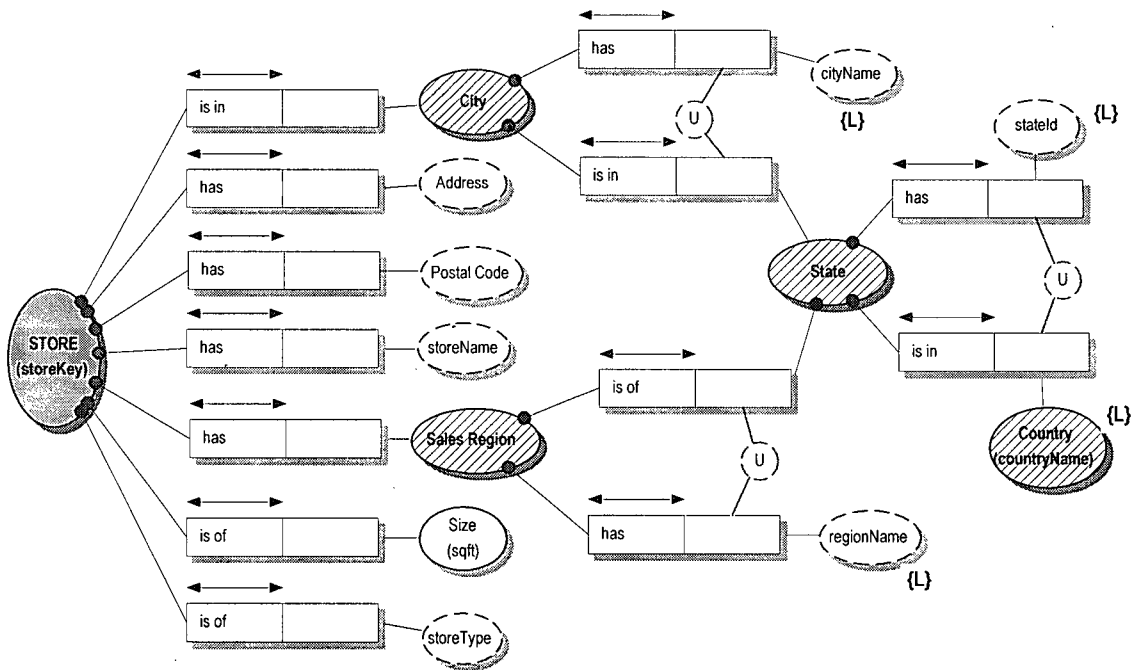


Figure B-6: MORM Level 4 - Store Dimension

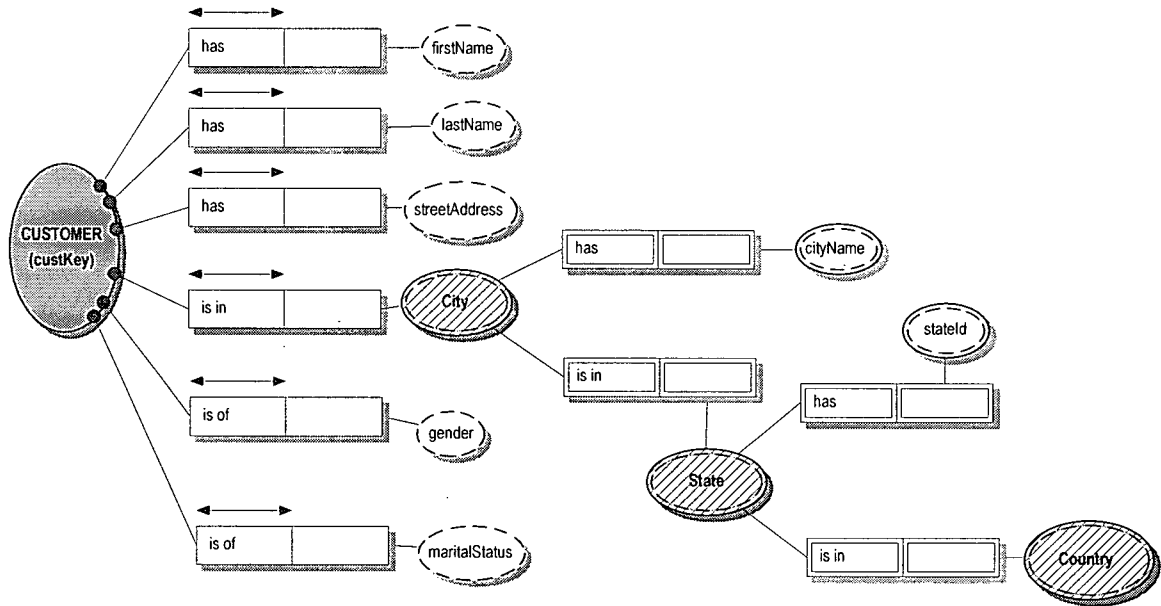


Figure B-7: ORM Level 4 - Customer Dimension

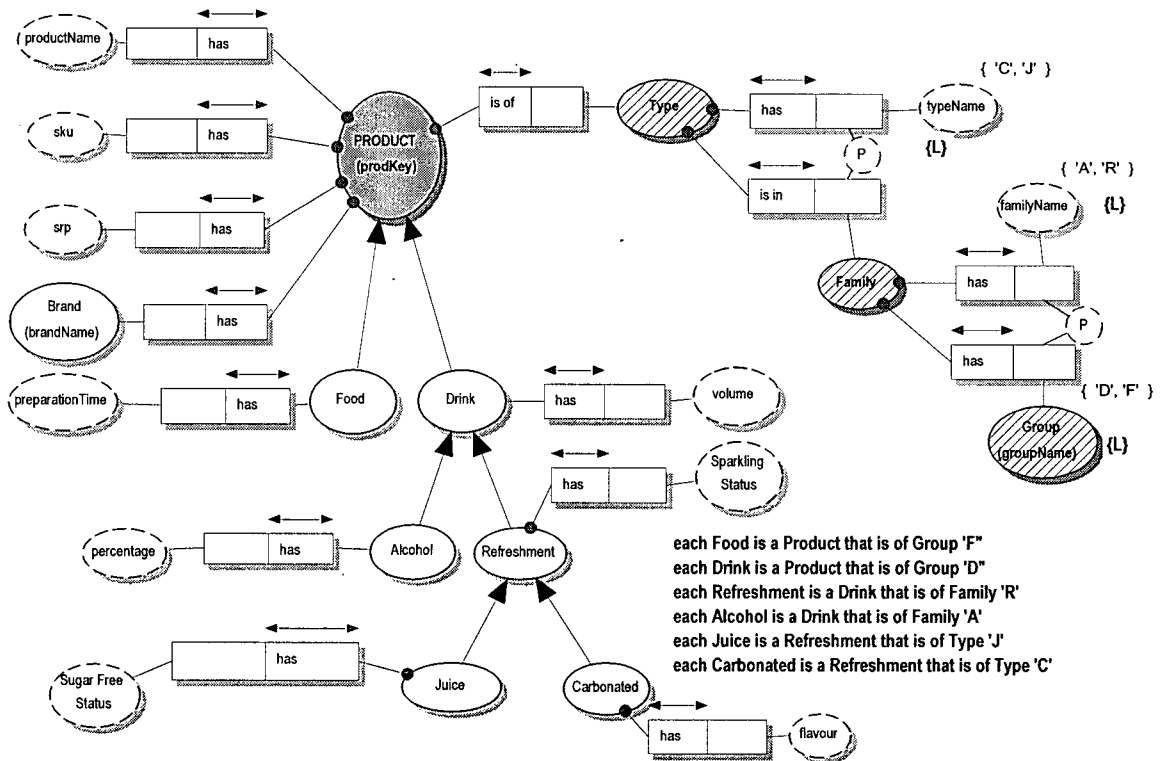


Figure B-8: ORM Level 4 - Product Dimension

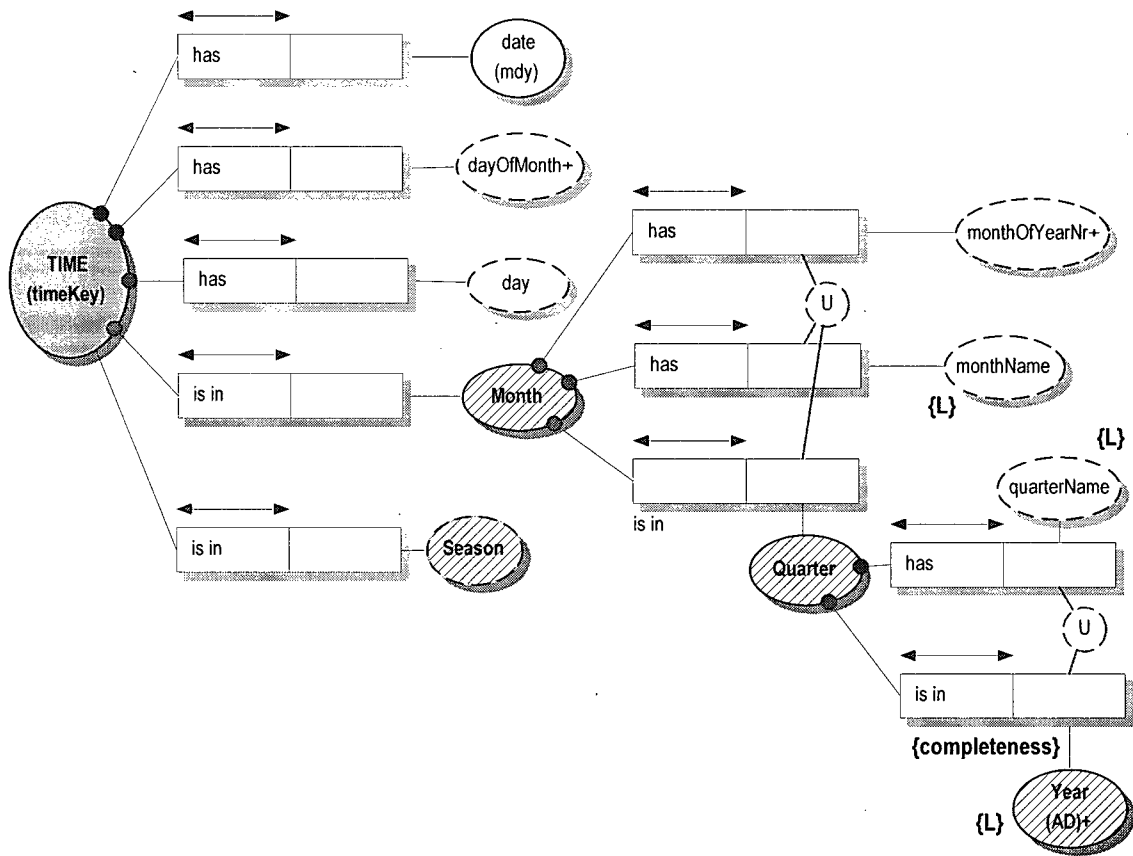


Figure B-9: MORM Level 4 - Time Dimension

## APPENDIX C: MORM IMPLEMENTATION RESULTS

Table C-1 summarizes how MORM constructs are implemented in our case study. Implementation details for each multidimensional requirement are included for the four stages of our implementation - conceptual, logical, physical, and OLAP. As presented in chapter 5, in the “Schema” column, conceptual refers to a *VisioModeler ORM diagram*, logical refers to a *VisioModeler dictionary document*, physical refers to a *SQL Server table* and OLAP refers to an *Analysis Services cube*.

Requirement	Schema	Implementation & Mapping Details
Business Process	Conceptual	Event object type connected to multiple Dimension object types via predicates.
	Logical	Star schema in graphical relational notation.
	Physical	Star schema tables in database.
	OLAP	Cube.
Business Process Families	Conceptual	Multiple Event object types connected to common “shared” Dimension object types.
	Logical	Multiple star schemas in graphical relational notation, central event tables linked to dimensions by foreign key connections.
	Physical	Multiple star schema tables in database, central event tables linked to dimensions by foreign key connections.
	OLAP	Multiple cubes with shared dimensions.
Event	Conceptual	Event object type.
	Logical	Central table of star schema in graphical relational notation.
	Physical	Central table of star schema in database.
	OLAP	Cube cell containing multiple quantitative measures that describe the event.
Atomic Measure	Conceptual	Entity or value type connected to Event object type via predicate.
	Logical	Event table column in graphical relational notation.
	Physical	Event table column in database.
	OLAP	Measure within cube.
Derived Measure	Conceptual	Fact type marked with an asterisk “*” to indicate its derivability, derivation rule included which references other fact types in the associated calculation. Double asterisk “**” indicates measure is to be stored in the physical database.
	Logical	Table column in graphical relational notation.
	Physical	Database table column.

Requirement	Schema	Implementation & Mapping Details
Additivity	OLAP	Measure within cube.
	Conceptual	Plus sign “+” following measure name indicates additivity, absence of “+” indicates non-additivity, semi additivity indicated by an informal rule as a comment in braces.
	Logical	Table column in graphical relational notation.
	Physical	Database table column.
Dimension	OLAP	Measures assumed additive by default, no support for semi-additivity.
	Conceptual	Dimension object type.
	Logical	Dimension table of star schema in graphical relational notation.
	Physical	Dimension table of star schema in database.
Dimension Attributes	OLAP	Dimension within cube.
	Conceptual	Entity or value type connected to Dimension object type via predicate.
	Logical	Dimension table column in graphical relational notation.
	Physical	Dimension table column in database.
Classification Hierarchies	OLAP	Member property of a dimension within a cube.
	Conceptual	Dimension object type forms the root of a dimension tree. Hierarchy object types represent each hierarchy level. A predicate between two hierarchy objects specifies a relationship between two hierarchy levels. Each level has an identifying label indicated with the constraint {L} next to its associated identifying value type object.
	Logical	Dimension table columns in graphical relational notation, all hierarchies are hidden (implicit) in flat dimension table.
	Physical	Dimension table columns in database, all hierarchies are hidden (implicit) in flat dimension table.
Multiple Hierarchies	OLAP	Hierarchy levels defined within cube.
	Conceptual	Hierarchy object types used to create different hierarchy paths from the root of a dimension.
	Logical	Dimension table columns in graphical relational notation, all hierarchies are hidden (implicit) in flat dimension table.
	Physical	Dimension table columns in database, all hierarchies are hidden (implicit) in flat dimension table.
Alternative Path Hierarchies	OLAP	Not explicitly supported. Defined as two or more dimensions with names that share the same dimension prefix but have different suffixes (e.g. Time.Calendar and Time.Season).
	Conceptual	Hierarchy object types used to create different hierarchy paths that converge into the same hierarchy level.
	Logical	Dimension table columns in graphical relational notation, all hierarchies are hidden (implicit) in flat dimension table.
	Physical	Dimension table columns in database, all hierarchies are hidden (implicit) in flat dimension table.
Shared Hierarchies	OLAP	Not explicitly supported. Defined as two or more dimensions duplicating the hierarchy beginning at the merging point.
	Conceptual	Object types and their predicates marked as <i>external</i> to indicate they are imported from another schema in which they are fully defined.
	Logical	Dimension table columns in graphical relational notation, all hierarchies are hidden (implicit) in flat dimension table.
	Physical	Dimension table columns in database, all hierarchies are hidden (implicit) in flat dimension table.
Shared Hierarchies	OLAP	Not explicitly supported. Defined as two or more separate dimensions starting at the dimension level where the merging occurs. Underlying dimension tables and aggregations are shared.

Requirement	Schema	Implementation & Mapping Details
Strictness	Conceptual	Many-to-one (n:1) uniqueness constraint on the first predicate role indicates each originating level is located in <i>at most one</i> target level. <i>Non-Strictness</i> indicated by a many-to-many (m:n) uniqueness constraint on the originating and target hierarchy level roles (e.g. sales region is comprised of more than one state, state is included in more than one sales region).
	Logical	No explicit support, all hierarchy information hidden in logical dimension table.
	Physical	No explicit support, all hierarchy information hidden in physical dimension table.
	OLAP	All hierarchies are strict by default as this is the only type supported, non-strict are converted to strict.
Completeness	Conceptual	<i>Completeness</i> indicated by defining the constraint {completeness} on the role of target hierarchy level. <i>Non-completeness</i> , indicated by absence of the constraint, all classification hierarchies are assumed non-complete by default.
	Logical	No explicit support, all hierarchy information hidden in logical dimension table.
	Physical	No explicit support, all hierarchy information hidden in physical dimension table.
	OLAP	No explicit support, implemented as strict hierarchy.
Categorization of Dimensions	Conceptual	Indicated by a generalization-specialization relationship to categorize entities that contain subtypes. Subtyping displayed using directed acyclic graphs, subtype nodes introduced when specific roles for them to play. Formal <i>subtype definitions</i> declared for all subtypes and written in the diagram.
	Logical	Dimension table columns in graphical relational notation, all hierarchies are hidden (implicit) in flat dimension table.
	Physical	Dimension table columns in database, all hierarchies are hidden (implicit) in flat dimension table.
	OLAP	Hierarchy levels defined within cube.
Degenerate Dimension	Conceptual	The constraint {DD} is placed next to the identifying object type associated with the Event.
	Logical	Event table column in graphical relational notation.
	Physical	Event table column in database.
	OLAP	Converted to separate dimension within cube, a single hierarchy level is defined for the identifier (e.g. ticket).
Many-to-Many Relationship Between Event and Dimension	Conceptual	Lower the event grain to the lowest dimension grain (e.g. product), resulting in multiple records (e.g. line items) for a single event (e.g. sale). Define additional uniqueness constraint for ticket and product, indicating their combination is unique. Include {DD} to identify degenerate dimension used to group records relating to a single sale event.
	Logical	Unique key column defined within event table (in graphical relational notation) to group multiple event records.
	Physical	Unique key column defined within event table (in database) to group multiple event records.
	OLAP	Degenerate dimension (e.g. ticket) defined to group multiple measures for a single event (e.g. multiple products per sale).

Table C-1: Details of ORM Implementation Results