Object-Relational Technology for Data Warehousing

by

Mr. Samit Bhakta

A Final Report of the Six-Credit Course
CS 6998 - CS 6999 System Development Project

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Computer Information Systems
Assumption University

November 2002

Object-Relational Technology for Data Warehousing

by
Mr. Samit Bhakta

A Final Report of the Six-Credit Course
CS 6998 – CS 6999 System Development Project

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Computer Information Systems
Assumption University

November 2002

Project Title	Object-Relational Technology for Data Warehousing
Name	Mr. Samit Bhakta
Project Advisor	Assoc.Prof.Dr. Suphamit Chittayasothorn
Academic Year	November 10, 2002

The Graduate School of Assumption University has approved this final report of the six-credit course, CS 6998 – CS 6999 System Development Project, submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Information Systems.

Approval Committee:

[illegible]
(Assoc.Prof.Dr. Suphamit Chittayasothorn)
Advisor

[illegible]
(Prof.Dr. Srisakdi Charmonman)
Chairman

[illegible]
(Air Marshal Dr. Chulit Meesajjee)
Dean and Co-advisor

[illegible]
(Asst.Prof.Dr. Vichit Avatchanakorn)
Member

[illegible]
(Assoc.Prof. Somchai Thayarnyong)
MUA Representative

November 10, 2002

ABSTRACT

Although data warehousing and object-relational database are no longer considered as recent technology, very limited work has been reported on object-relational data warehousing. Most data warehouses are built on relational databases. This study attempts to develop a warehouse data model, which could be implemented using an object-relational DBMS.

It has been found out that the object-relational database can support an improved data warehouse that might not be achieved using relational or object-oriented databases.

Object-relational technology is capable of capturing and handling the slowly changing attributes of entities by using complex data structures involving time, which would not be possible using a relational database. Handling slowly changing attributes remains a major issue in data warehousing.

On the other hand, it is possible to eliminate the major drawback of object-oriented databases – the inability to formulate query using the SQL language. Object-relational databases support the SQL3 language, which is similar to the popular SQL92 language, and therefore, the task of formulating queries remains easy.

Despite those advantages, non-standardization of SQL3 language remains one of the major obstacles in commercially implementing the object-relational technology.

ACKNOWLEDGEMENTS

It is a great pleasure for the writer to acknowledge the assistance of a large number of individuals to this project.

First of all, the writer would specially thank Associate Professor Dr. Suphamit Chittayasothorn, his advisor for this project, for his valuable guidance and advice, without which this work would not have been possible. The quality and depth of instruction in his database classes and his active involvement in this project have been the key to its success.

The writer would also like to extend his thanks to all his instructors of Assumption University for their valuable instruction in the course of pursuing his degree.

The writer acknowledges the support of his classmates for their continuous encouragement. In particular, the writer recognizes the effort of James C. Baldwin, for suggesting various improvements in this project. Of course, any mistakes should be attributed to the writer alone.

Last, but not least, the writer gratefully acknowledges the support, encouragement, and patience of his family, without forgetting little Ankita.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

1.1 Overview

The concept of data warehousing has been evolving for quite a long time. In the eighties, a single database used to serve both operational and analytical needs. Databases were designed for the transaction processing system. Data analysis was based mostly on human interactions by extracting data from a transaction processing database. This was acceptable to the organizations when the cost of hardware and software was too high to maintain a separate database for analytical purposes and the number of records to deal with was limited. Also, business had yet to realize the value of analyzing data in the service of crucial decision-making.

However, there is a limited technology available today to optimize a database for both transaction processing and analytical requirements, though their processing characteristics are fundamentally different.

A transaction processing database deals with present data, which needs to be updated frequently. The presence of a vast amount of historical data degrades the performance of an operational database. Redundancy must be kept at a minimum level, as it causes insert, update, and delete anomalies.

On the other hand, data analysis is based on historical data, which are read-only in nature. Therefore a warehouse database must be able to deal with a huge amount of data, which are only rarely updated. Redundancy is not a major issue for a data warehouse if it improves the query performance.

However, based on the above-mentioned aspects of transaction processing and warehouse databases, it is not impossible to design a database to serve the purposes, particularly if the volume of data involved is not too large. What makes data warehousing indispensable is that it integrates the data scattered around the business in different forms e.g. relational database files, spreadsheets, flat databases, and common text files. Data warehousing extracts information from the different heterogeneous types of databases in an organization and places them in a common database structure so that the user community is able to access all data from a single database.

Therefore, a warehouse database uses a completely different approach to organizing the data even though the source of the data is the transaction processing database. The data warehouse extracts data from the transaction processing system, cleans it, and maps it to a different schema, which can deal with large data volumes and provide good query performance.

There are different data models to satisfy these data warehouse needs. Among them the star schema, which uses de-normalized tables, and the starflake schema, which is a variation of the star schema that selectively normalizes some dimension tables, are popular and widely used.

An Object-Relational Database Management System (ORDBMS) is considered a hybrid, combining the concept of Object-Oriented Database Management Systems (OODBMS) to deal with complex and user-defined data structures and the strength of Relational Database Management Systems (RDBMS), i.e. the simplicity of data representation at the conceptual level in tabular form and the support of a query language for data definition and manipulation. According to Ullman et al. 1999, support of non-atomic attributes, reference pointers, abstract data type, and user defined methods within tabular data representation made object-relational database unique from other existing database technologies.

Recent literatures on data warehousing suggest that current research focuses mostly on relational technology, although a few use OODBMS. This may not be the case at present when major DBMS have already incorporated object-relational features.

The use of ORDBMS to build a warehouse database could result in an improved warehouse. This study attempts to discover potential opportunities for improvement of warehouse databases using ORDBMS.

1.2 Objectives

The objective of this project is to examine the effects of some of the object-relational features of ORDBMS on data warehousing, which is traditionally built on relational databases. This project attempts to investigate the possibility of improvement if an ORDBMS is used for a warehouse database. The issue of slowly changing attributes is also investigated in the light of object-relational technology. The study is limited to the conceptual level in the form of data definition and manipulation using structured query language. Performance implications are considered to be out of the scope of the current study.

1.3 Scope of Work

To study the impact of object-relational technology on data warehousing, a case study was undertaken to build a warehouse database incorporating some features of object-relational technology. A hypothetical database was considered for a transaction processing system, based on which the warehouse data model was developed using the star schema data model. This database schema was then implemented using an ORDBMS, Oracle 8i. The strengths and drawbacks of object-relational technology were then measured based on the implementation.

1.4 Organization of This Report

This report is divided into seven chapters. This chapter, Chapter I, presents an introduction to the project. Chapter II reviews the recent literature on data warehousing. Chapter III provides a close look at object-relational technology – its evaluation and its advantages over relational and object-oriented databases. In Chapter IV, a data warehouse schema is developed. This schema is implemented in Chapter V using Oracle 8i. Chapter VI illustrates the results and presents a discussion based on the schema developed and its implementation. Finally, Chapter VII contains the conclusions of this study and the scope of further work in this field.

II. LITERATURE REVIEW ON DATA WAREHOUSING

Study of the recent literature suggested that very little work has been done to build data warehouse on object-relational technology. Surprisingly, not a single published article was found proposing object-relational data model for data warehousing, despite the fact that object-relational DBMS has been commercially available for more than four years now.

This chapter is broadly classified into two sections. Section 2.1 generally reviews data warehousing and its benefits. Reviews are also conducted in this section on how a warehouse database differs from a transaction processing database. Section 2.2 reviews the different types of data models available for warehouse databases, and discusses their strengths and weaknesses.

2.1 Data Warehousing and Its Benefits

A data warehouse contains information that is collected from individual data sources and integrated into a common repository for efficient query and analysis. The concept of data warehousing was first proposed by Inmon and Kelley 1993 (Chen et al. 1999).

An operational or transaction database system helps to run the operations of the organization using conventional methods. However, it has become increasingly obvious that these legacy systems are unable to handle information management functions such as planning, forecasting, and financial analysis. The structure of operational data is often complex as the data is highly normalized and it is not always meaningfully presented to the end user (Chelluri and Kumar 2001).

On the other hand, data stored in a data warehouse is historical in nature, which provides the analyst with information for analyzing the past performance of the organization and plans for future activities. To achieve this, the analyst needs immediate access to the terabytes of information stored in heterogeneous databases. According to Hanson 1997, there are two primary focuses of effort as to how information from heterogeneous databases should be integrated to provide the user community with the ability to access this information through a single access method. The first method involves the creation of either a relational, or an object-oriented Multidatabase System structure that contains the necessary information to access the underlying, or source databases. The second method involves extracting the information, which could be the entire database, and placing it in a database structure that is available to the user community. The second methodology is the one being primarily implemented by the commercial vendor community under the title of data warehousing.

Even if there is a single database used for transaction processing system, using only one database for both online transaction processing and decision support concurrently degrades the performance of the operational system and user response time (Chelluri and Kumar 2001).

Furlow 2001 described the five principle benefits of data warehousing. The first, and most obvious, is that data warehousing simplifies decision making because it provides a single view of the data by pulling it together from disparate and potentially incompatible locations throughout the organization. The second benefit is easy and quick access to data, leading to increased productivity and efficiency gains. Increased performance of operational systems is the third benefit of warehousing, since queries generated by the user do not interfere with normal operations. The fourth benefit is the flexibility and the scalability of data warehouses since they give an organization a computing infrastructure that can support change in both the computer system and the business structure. Finally, with a data warehouse in place, an organization can better manage and use its knowledge, which in turn helps the organization become more competitive, better understand its customers, and more rapidly meet the market demand.

Furlow 2001 has also pointed out the potential disadvantages of data warehousing, which are the complexity and anticipation in development, making data warehousing project costly and time consuming. An iterative development approach starting with a pilot system could be the key to a successful data warehousing project.

2.2 Warehousing Data Models

Kimball 1997 described Dimensional Modeling (DM), a logical design technique, for data warehousing. DM seeks to present the data in a standard, intuitive framework that allows for high-performance access. It is inherently dimensional, and it adheres to a discipline that uses the relational model with some important restrictions. Every dimensional model is composed of one table with a multipart key called the fact table, and a set of smaller tables called dimension tables. Each dimension table has a single-part primary key that corresponds exactly to one of the components of the multipart key in the fact table. This characteristic "star-like" structure is often called a star join. Due to this, the dimensional data model is also called a star schema. Kimball 1997 described many benefits of DM for warehouse data modeling, among which flexibility, adaptability, expandability, and standardization are prominent.

An alternative to the dimensional data model described above is the multidimensional data model described by Agrawal et al. 1997. This model is based on the "hypercube" paradigm. Here, the business measures, i.e. facts, are stored within the cells of a hypercube whose axes or edges represent business entities or attributes, which are also called dimensions.

Although the above data models, i.e. dimensional modeling and multidimensional data modeling, are structurally similar, they are implemented using different technologies. Dimensional modeling uses a relational database, whereas multidimensional data modeling uses a multidimensional database. Trujillo et al. 2001 proposed the use of multidimensional modeling using an object-oriented database. However, Giovinazzo 2000, preferred a relational-based dimensional modeling in implementing the database, as many cells of the multidimensional database remain empty which wastes a significant amount of storage space.

Sarda 2000 pointed out that changes to a dimension are a major issue for data warehousing which is not adequately addressed either in the hypercube-based multidimensional model or in star schema based dimensional modeling. Kimball 1996 proposed three different techniques to deal with slowly changing dimension attributes. Giovinazzo 2000 proposed two additional possibilities. Incidentally, as explained in Chapter VI, all of them were using the relational framework. Sarda 2000 suggested two orthogonal time measures, called valid (real-world) time and transaction (system) time, to maintain the complete history of entities. This paper shows that the dimensions and facts can be modeled as temporal states or event relations to ensure the consistency of data in a data warehouse. This idea was basically derived from the temporal database concept. However, this paper does not specify the implementation technology for this logical model.

Chelluri and Kumar 2001 proposed a multidimensional grid frame organization to deal with very large data warehouses. This helps to manage the size of the warehouse database using a kind of garbage collection scheme, which categorizes the data warehouse components and relocates them to specific locations. Thus, active data and stale reside at different places, which improves the usefulness of the warehouse.

III. OBJECT-RELATIONAL DBMS

In the commercial world of Database Management Systems (DBMS), there are several families of DBMS products available. Two of the most dominant ones are Relational DBMS (RDBMS) and Object-Oriented DBMS (OODBMS). Other types of DBMS products, which are based on the hierarchical and network data models, are now being referred as legacy DBMS (Elmasri and Navathe 2000).

During the past few years, several major software companies including IBM, Informix, Oracle, and Sybase have all released Object-Relational versions of their products. These companies are promoting these new, extended versions of relational database technology as Object-Relational DBMS, also known as ORDBMS. The main forces behind the development of ORDBMS stem from the limitation of the legacy and basic relational DBMS, which are unable to meet the challenge of non-traditional new applications such as storing images and multimedia objects in the database. Consequently, the objects and related operations are becoming more complex. Initially OODBMS looked promising to handle the objects and related operations such as images, geographical information systems, multimedia objects, 3-D, and temporal data. However, they have been unable to live up to the expectation. This leads to the development of new technology, ORDBMS, which is a combination of both relational and object-oriented concepts. The main advantages of ORDBMS are massive scalability and support for object-oriented features.

This chapter compares and contrasts this new class of database (ORDBMS) with the relational databases (RDBMS) from which they are evolving and also with efficient object-oriented databases (OODBMS). The concepts of RDBMS and OODBMS are discussed in Section 3.1 and 3.2. In Section 3.3, the concept of ORDBMS is presented and the advantages of ORDBMS over RDBMS and OODBMS are discussed.

3.1 Relational DBMS (RDBMS)

The relational model was formally introduced by Dr. E. F. Codd in 1970 and has evolved since then, through a series of writings and later through implementations by IBM, Oracle, and others. The defining standard for relational databases is published by ANSI (the American National Standard Institute) as SQL (ANSI 1986) or SQL1, also called SQL-86. A revised standard is called SQL2, also referred to as SQL-92.

A relational database is composed of relations in the form of two-dimensional tables of rows (tuples) and columns. Organizing data into tables, in which form data is presented to the user and the programmer, is known as the logical view of the database. The stored data on a computer disk system is called the internal view. The tuples are called records and the columns (fields in the record) are called attributes. Each column has a data-type (i.e., integer, float, date). There are various restrictions on the data that can be stored in a relational database. These are called constraints. The constraints include domain constraints, key constraints, entity integrity constraints, and referential integrity constraints. These constraints ensure that there are no ambiguous tuples in the database.

RDBMS use Structured Query Language (SQL, currently SQL2) as the Data Definition Language (DDL) as well as the Data Manipulation Language (DML). SQL includes statements for data definition, modification, querying, and constraint specification. The types of queries vary from simple single-table queries to complicated multi-table queries involving joins, nesting, set union / differences, and others. All processing is based on values in fields of records. Examples of RDBMS include Oracle, developed by Oracle Corporation, DB2, developed by IBM, and Microsoft SQL Server developed by Microsoft. The SQL standard enables users to easily migrate their database applications across database systems. In addition, users can access data stored in two or more RDBMS without changing the database sub-language (SQL). The other merits include rapid data access and large storage capacity. The main disadvantages of Relational Databases include their inability to handle application areas like spatial databases (e.g. CAD), applications involving images, special types in databases (e.g. complex numbers, arrays, etc.) and other applications that involve complex interrelationships of data (Elmasri and Navathe 2000).

3.2 Object-Oriented DBMS (OODBMS)

The desire to represent complex objects in a way that facilitates Object-Oriented (OO) systems design, has led to the development of OODBMS. The concept of abstract data-types (ADT) evolved, in which the internal data structure is hidden and the external operations can be applied on the object that is specified, led to the concept of encapsulation. The main features of OO programming languages are encapsulation, inheritance and polymorphism (Bahrami 1999). Encapsulation can be thought of as a protective layer that prevents the code and the data from being accessed by other code defined outside the layer. The process in which one object inherits the properties of a previously defined object is called inheritance. Inheritance aids in the reuse of existing definitions for creating new objects. Polymorphism allows the same operator or symbol to have different implementations, depending on the types of objects to which the operator is applied.

OO databases employ a data model that supports the object-oriented features discussed above as well as abstract data-types (Date 1999). OO databases provide unique Object Identifier (OID) so that the objects can be uniquely identified. This is similar to a primary key in the relational model. OO databases utilize the power of OO programming languages to provide excellent database programming capability. The data in OODBMS is managed through two sets of relations, one describing the interrelations of data items and another describing the abstract relationships (inheritance). These systems employ both relation types to couple data items with procedural methods (encapsulation). As a result, a direct relationship is established between the application data model and the database data model. The strong connection between application and database results in more compact code, more natural data structures, and better maintainability and reusability of code. OO languages, such as C++ or Java, are able to reduce code size by not having to translate code into a database sublanguage such as SQL and ODBC or JDBC.

The main drawback of OODBMS has been poor performance. Unlike RDBMS, query optimization for OODBMS is highly complex. OODBMS also suffer from problems of scalability, and are unable to support large-scale systems. Some examples of OODBMS are Ardent (formerly O2) developed by Ardent Software, and the ObjectStore system produced by Object Design Inc. (Elmasri and Navathe 2000).

3.3 Object-Relational DBMS (ORDBMS)

The main objective of ORDBMS design was to achieve the benefits of both the relational and the object models such as scalability and support for rich data-types. ORDBMS employ a data model that attempts to incorporate OO features into RDBMS. All database information is stored in tables, but some of the tabular entries may have richer data structure, termed abstract data-types (ADT). An ORDBMS supports an extended form of SQL called SQL3 that is still in the development stages and, hence, not standardized. Extensions are needed because ORDBMS have to support ADT's. The ORDBMS subsumes relational model because the data is stored in the form of tables having rows and columns, because SQL is used as the query language and because the result of a query is also a table or a set of tuples. But the relational model has to be drastically modified in order to support the classic features of OO programming.

According to Stonebraker and Brown 1999, the principle characteristics of ORDBMS are (1) base data-type extension, (2) support for complex objects, (3) inheritance, and (4) implementation of a rule system.

ORDBMS allow users to define data-types, functions and operators. As a result, the functionality of the ORDBMS increases along with their performance.

The following is an example schema of an employee relation, which ORDBMS supports:

EMPLOYEE (ID, Name (last, first, middle), Sex, BirthYear, Address (Street, City, State, Zip), Phones (Location, Number), DepartName, DateOfHire, Salary, Picture)

This is to be noted that "Name" and "Address" are composite attributes, "Phones" is a combination of multivalued and composite attributes, and "picture" is an image. None of the above is supported by traditional RDBMS technology.

Stonebraker and Brown 1999 have classified the DBMS applications into four types – simple data without query, simple data with query, complex data without query, and complex data with query. These four types describe file systems, Relational DBMS, Object-Oriented DBMS, and Object-Relational DBMS, respectively. The current ORDBMS in commercial market include Oracle9i from the Oracle Corporation, DB2 from IBM, and Universal Server by Informix (recently taken over by IBM). Stonebraker and Brown 1999 predicted that applications from Relational DBMS (simple data with query) would slowly move towards the Object-Relational DBMS (complex data with query).

The main advantage of ORDBMS is their massive scalability. Oracle9i, released by the Oracle Corporation, is designed to manage large amounts of information. In spite of their many advantages, ORDBMS also have a drawback. The architecture of object-relational models is not appropriate for high-speed web applications. However, with advantages like large storage capacity, access speed, and the manipulation power of object-databases, ORDBMS has conquered the database market. The support from major DBMS vendors and its features have made ORDBMS the market leader.

IV. OBJECT-RELATIONAL DATA WAREHOUSING

The strengths of ORDBMS over RDBMS are discussed in the last chapter. Considering the present trend in database technology and support by leading database vendors, new generation data warehousing should take the advantage of ORDBMS. In this chapter, a data warehouse schema will be presented and discussed using object-relational technology. To make the things simple, the warehouse database schema will be based on the popular star schema data model of data warehousing.

In Section 4.1, an Entity Relationship (ER) Diagram of a transaction processing database is presented on which basis the data warehousing schema was developed. The warehousing database is presented in Section 4.2. Section 4.3 proposes a modification of the data warehousing schema described in Section 4.2 by tuning it in accordance to the needs of the practical requirements of the specific application. Finally, in Section 4.4 a conclusion is drawn based on the presented data warehouse schemas. It is worth mentioning here that the schemas presented in this section are application independent. The implementation of the schema is considered in the next chapter.

4.1 ER Diagram of Transaction Processing Database

To study the effect of object-relational technology on data warehousing, the case of a car dealer's sales transaction processing system is considered. The car dealer records the sales transactions using a transaction processing database. As the study concentrates on data warehousing, discussion of this transaction database is limited to describing the schema only, without detailing how the schema is achieved. Further, the transaction database is considered as an object-relational type as well, which allows multivalued and composite attributes although these do not satisfy the first normal form requirements of relational technology.

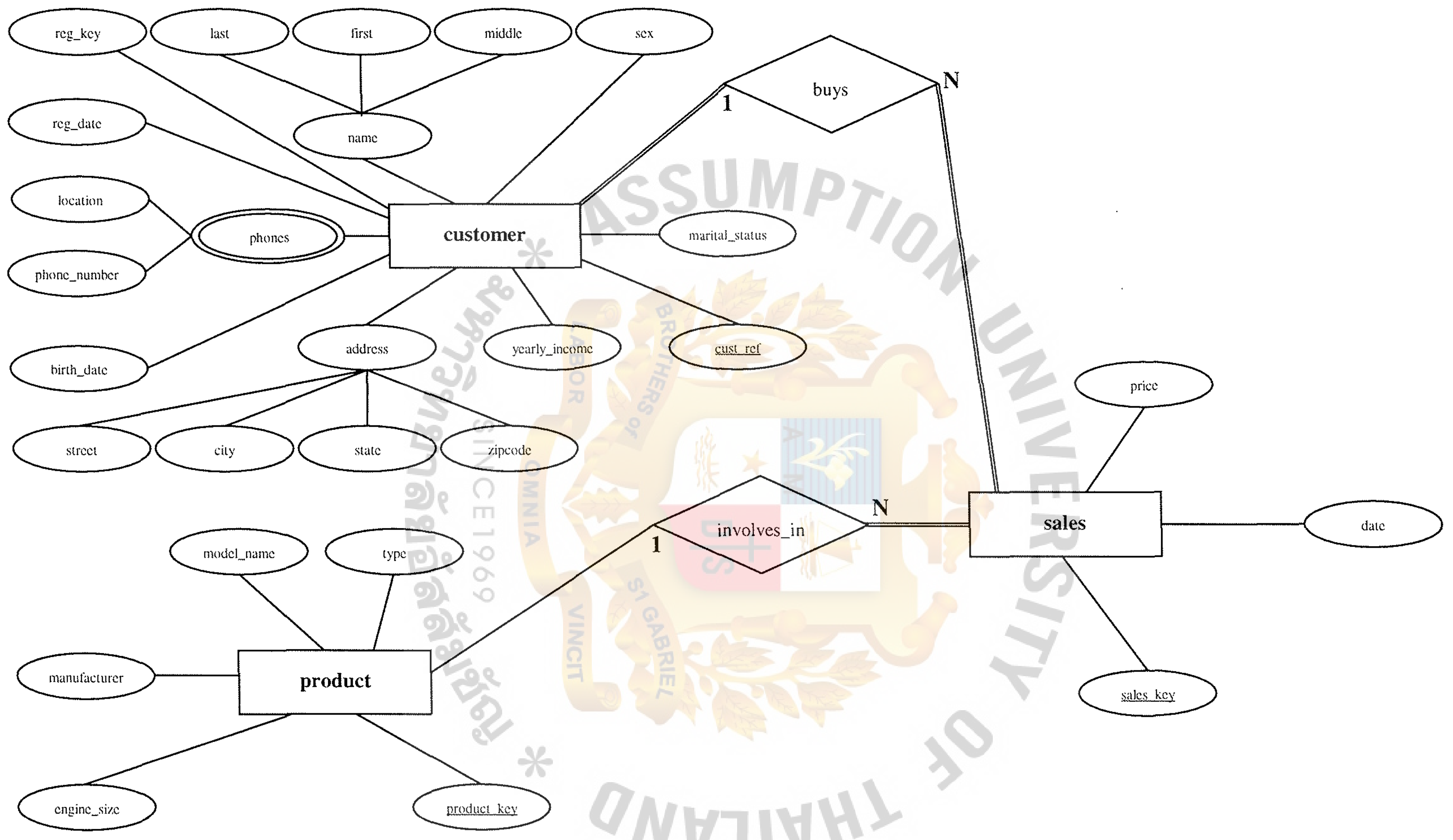

Figure 4.1. ER Diagram of Transaction Processing Database.

Figure 4.1 represents an ER Diagram of the transaction database based on Peter Chen's classical entity relationship model. It consists of three entities – "customer", "product", and "sales". Entity type "customer" holds data about the customer. It has few simple attributes like "reg_key" (a unique key serving as the key attribute of the customer entity type) "sex" (customer's sex, e.g. male or female), "birth_date" (customer's date of birth), "marital_status" (customer's marital status, e.g. unmarried, married, separated etc.), and "yearly_income" (customer's yearly income). Entity type "customer" has a composite attribute like "name" (customer's name), which consists of three simple attributes – "last" (customer's last name), "first" (customer's first name), and "middle" (customer's middle name). Entity type "customer" also has another composite attribute "address" (customer's address), which consists of four simple attributes – "street" (customer's street name), "city" (customer's city), "state" (customer's state), and "zipcode" (customer's zipcode). Attribute "address" is not considered as a multivalued attribute, and therefore, only one address (e.g. home or office) per customer can be registered at a time. Entity type "customer" also has a multivalued composite attribute called "phones". Multiple sets of "location" (location of a phone number e.g., home, office) and "phone_number" (phone number associated with the location) of a customer can be registered under "phones" attribute.

Other than those mentioned above, the entity type "customer" has two more simple attributes – "reg_date" and "cust_ref" which require some special attention. It is assumed that the customer needs separate registration every time a new car is purchased. The "reg_date" (registration date) simply records the system date on the date of registration. The attribute "cust_ref" (customer reference) is to identify a customer who has registered multiple times while purchasing several new cars. As "reg_key" is a key attribute of the "customer" entity type, it is unable to identify the uniqueness of a customer in case of multiple registrations. Attribute “cust_ref” resolves this issue. A “cust_ref” is allotted during the first registration of a customer. On subsequent registrations, the “cust_ref” remains unchanged. This also assists the data warehouse schema to cope up with the slowly changing attributes of the customer. In the next section, this is discussed more elaborately.

As it is considered that the schema will be implemented using object-relational technology, technically there is no need of having a key attribute (like “customer_key” in case of “customer” entity type) for entities. Each tuple of a table will be allotted a unique OID (Object Identifier) by the ORDBMS, which could be served as primary key. However, there is no harm in keeping a primary key, as it would assist readers familiar with the relational database.

Entity type “product” records data about a product (car). It consists of simple five attributes – “product_key” (a key attribute identifying each tuple of the “product” table uniquely), “model_name” (model name, the name of the model of the product), “manufacturer” (manufacturer of the product), “type” (describes the type of the product, e.g. 4 door sedan), and “engine_size” (engine size, describing the size of the engine, e.g. 2000CC). Attribute “product_key” is optional for the reasons mentioned above.

Entity type “sales” describes the sales transaction. It has three simple attributes – “sales_key” (key attribute of the “sales” entity type), “date” (transaction date which will be same as system date), and “price” (price paid for the product). Moreover, entity type “sales” have a many-to-one relationship with the “customer” and “product” entity types. Due to this, the “sales” table shall have two foreign keys “customer_key” related to the “customer” entity type, and “product_key” related to the “product” entity type. For object-relational technology, the conventional primary / foreign key relationship can be more effectively implemented using pointers. Therefore, each tuple of “sales” table requires two pointers pointing the associated tuples of the "customer" and "product" tables.

The schema for the sales transaction database is a basic one and may not seem to be very practical. The schema is kept simple intentionally, so that the points associated with the objective of this study could be highlighted. The objective of this study is to examine the implementation of object-relational technology in data warehousing and not to develop a database for a car dealer information system.

4.2 ER Diagram of Warehousing Database

Based on the transaction processing database described in Section 4.1, the ER diagram of warehousing database is described in this section. The warehousing database is based on simple star schema data model, indicating that there are two basic types of tables – dimension and fact. The fact table is the center table which contains the facts, i.e. numbers that describe the characteristics of the relationship between the dimensions. Dimension tables surround the fact table and the primary keys of the dimension table are foreign keys of fact table.

Unlike a transaction processing database, where all tables need to be normalized to reduce redundancy, the dimension tables of a star schema data model are denormalized which improves performance and simplifies the queries. This makes sense, as redundancy and data integrity are not a major issue in data warehousing, which basically consists of read-only data.

The basic purpose of data warehousing is to analyze historical data. Due to this, it is not necessary to have all the attributes of a transaction processing database mapped in the warehouse database. For example, in this case study, it is unlikely that the name, street address and phone numbers of the customer will be queried. Therefore, these attributes need not be a part of the dimension table. However, as demonstrated by

Giovinazzo 2000, some of those data, which are of a nonanalytical type, could be a part of a shadow dimension table, which could be retrieved on the rare occasion when a business question demands it. A shadow dimension table is not a true dimension in the sense that it binds the analysis space. The shadow dimension hardly provides additional information concerning the dimension required for data analysis. The shadow dimension can be joined to the dimension table it shadows in a one-to-one relationship. It may also be joined to the fact table in a one-to-many relationship.

Following the preceding discussion, Figure 4.2 represents an ER Diagram for a warehouse database corresponding to the ER Diagram of the transaction database described in Figure 4.2. In this diagram, the "customer_dimension", "product_dimension", and "time_dimension" entity types form the dimension tables, whereas the entity type "sales_fact" forms the central fact table binding the dimensions together. The "customer_shadow_dimension" entity type, which contains nonanalytical data of customer dimension, is joined to the "customer_dimension" in a one-to-one relationship. Data of the "customer_dimension" and "customer_shadow_dimension" tables are to be mapped from the "customer" table of transaction database, whereas data of "product_dimension" and "sales_fact" are to be mapped from the "product" and "sales" tables respectively.

The "customer_dimension" entity type is designed to handle slowly changing attributes of customer, which may affect the data analysis. For example, the shopping behavior of customer may change with the marital status. When an unmarried customer purchases a car the customer data is registered. A few years later, the same customer purchases another car after being married and the customer's new marital status is registered again. In the warehouse database, if the customer's new record replaces the old one, an analysis on purchasing behavior of customers based on marital status may produce a misleading result.

It depends on the application and user requirement whether the slowly changing attributes of the entity are to be captured. One of the objectives of this study is to examine the issue of slowly changing attributes of dimension entities. For this purpose, four attributes of the "customer_dimension" entity type are assumed to be slowly changing. Those are "marital_status", "yearly_income", "city", and "state". Each of those changing attributes is considered to be multivalued composite type, which is supported by object-relational technology. They store the attribute values on the date of registration and the validity period of the values, i.e. from when to when were the attributes valid.

For example, if the case of marital status is considered, when an unmarried customer is registered for the first time, the "marital_status" value is set to unmarried, "from_date_key" is set to the date of registration, and "to_date_key" is set to maximum (the maximum possible date value permitted by the DBMS). Next time, when the same customer purchases a second car after being married, a new set of value is appended by setting the "marital_status" value to married, "from_date_key" is set to the date of second registration, and "to_date_key" is set to infinity. Also, the "to_date_key" value of the first set of records is set at the second registration date. Therefore "unmarried" was valid from the first registration date to the second registration date, whereas "married" is valid from the second registration date to the maximum. If the marital status remains unchanged at the time of the second purchase, update of marital status is not required. With this kind of data structure, it is possible to identify the exact value of the attributes of the customer at the time of purchase, and thus accurate analysis would be possible.

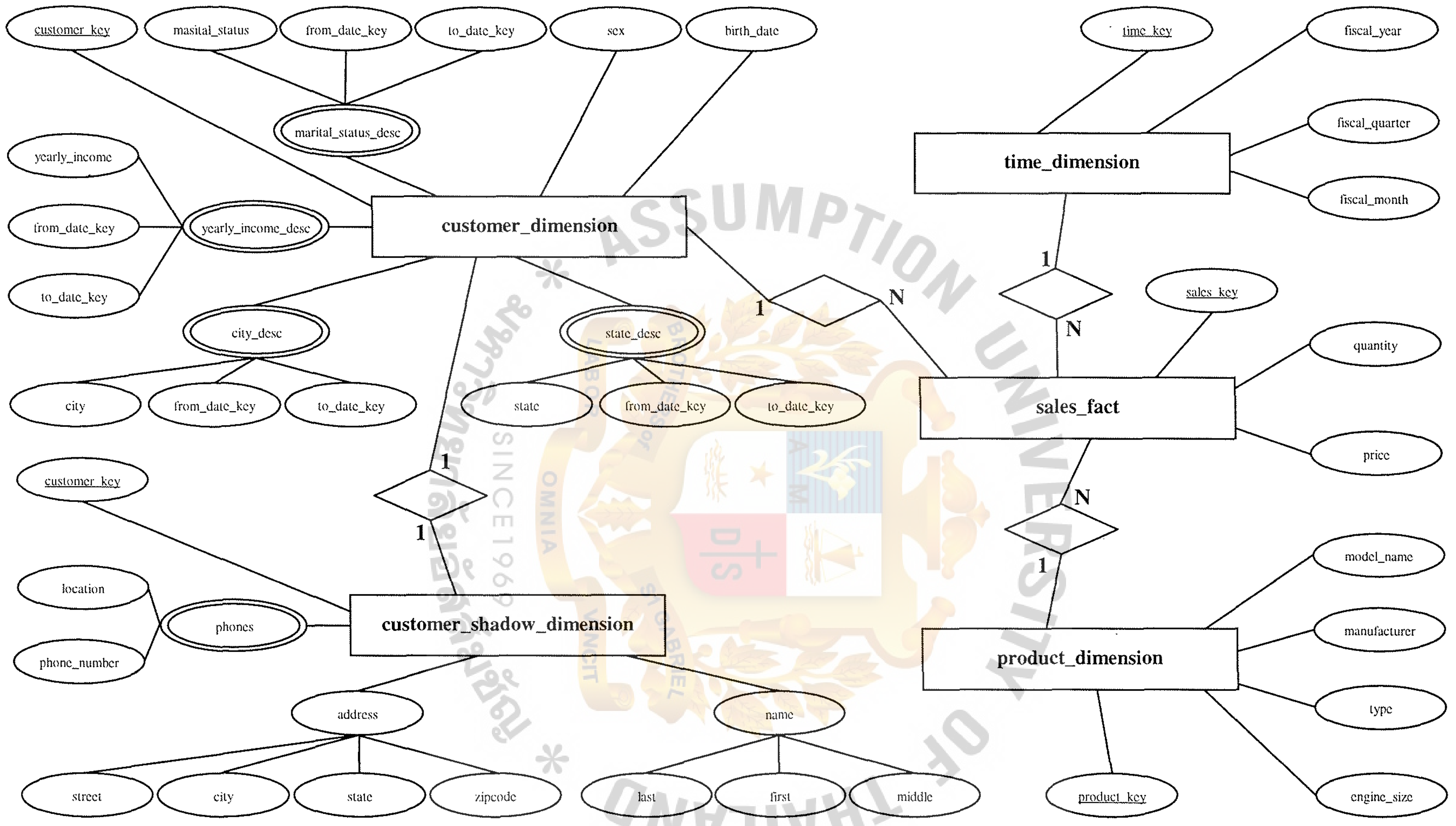

Figure 4.2. ER Diagram of Warehouse Database.

The attributes "cust_ref" and "reg_date" of entity type "customer" of the transaction database assist in this kind of mapping from transaction to warehouse database. Attribute "cust_ref" keeps track of customer's uniqueness, and "reg_date" assists mapping the "from_date_key" and "to_date_key" values.

The "customer_dimension" entity type has few simple attributes like "customer_key" (the key attribute of the "customer_dimension" entity type), "sex" (the customer's sex, e.g. Male or Female), and "birth_date" (the customer's date of birth). These simple attributes are considered nonchanging, signifying that the "sex" or "birth_date" does not change over time. Other attributes of "customer_dimension" are multivalued composite attributes and are considered to be changeable over time. Those are "marital_status_desc" (material status description, keeping track of the customer's marital status on the date of purchase), "yearly_income_desc" (yearly income description in $, keeping track of the customer's yearly income on the date of purchase), "city_desc" (city description, keeping track of the city where the customer resides on the date of purchase), and "state_desc" (state description, keeping track of the state where the customer resides on the date of purchase). Each of these slowly changeable attributes are multivalued in nature and consists of three simple attributes – the name of the attribute itself (like "marital_status" or "yearly_income") which holds the variable value, and "from_date_key" and "to_date_key" attributes representing the period of time for which the attribute was valid. The attributes "from_date_key" and "to_date_key" are considered foreign keys of the "time_dimension" entity type (described in the next paragraph) based on the key attribute "time_key". With this arrangement, it is possible to answer a time-related query involving slowly changing attributes.

In this case study, for the warehouse database, the granularity of the time is considered as one month. The entity type "time_dimension" serves to filter the facts with respect to time for time-related queries, which are frequent in reality. It has four simple attributes – "time_key" (the key attribute of "time_dimension" entity type), "fiscal_year" (fiscal year), "fiscal_quarter" (fiscal quarter), and fiscal_month (fiscal month).

The entity type "product_dimension" is similar to the "product" entity type of the transaction database. All of the attributes remain intact, as business queries are expected on any of those attributes.

The center of the star schema is the "sales_fact" entity type. It consists of a key attribute "sales_key", and two other simple attributes, "price" (indicating the price of the car) and "quantity" (indicating the quantity of cars sold). Attribute "quantity" is a derived one (the transaction database records sales of each car in separate tuples), but there is no harm in having it in warehouse database. It may help in the formulation of "number of cars sold" related queries which is quite common. The entity type "sales_fact" has a many-to-one relationship with each of the dimension tables, indicating that the "sales_fact" table shall have three foreign keys, each of which are a primary key of a dimension table – "customer_key" from the "customer_dimension" table, "time_key" from the "time_dimension" table, and "product_key" from the "product_dimension" table. Although the combination of those foreign keys is also unique, a separate primary key "sales_key" is provided as recommended by Giovinazzo 2000.

The last entity type considered in warehouse database is the "customer_shadow_dimension". As described before, this entity does not contain analytical data. This entity type is joined to the "customer_dimension" entity type with a one-to-one relationship. The entity type “customer_shadow_dimension” contains only the latest information about the customer, such as name, address, and telephone numbers. They are mapped directly from the “customer” table of the transaction database. It has two composite attributes – “name” (consisting of “last”, “first”, and “middle”) and “address” (consisting of “street”, city”, “state”, and “zipcode”). It also has a multivalued composite attribute “phones” which is identical to the one discussed under the transaction processing database in Section 4.1.

This warehouse schema was implemented using an ORDBMS and tested against standard business queries. The result was found satisfactory and was able to answer general queries with ease. However, some difficulties were experienced in formulating time-related queries involving slowly changing attributes.

To solve a query such as finding car sales to unmarried customers during some year, it is necessary to find the intersection of the sets of the sales fact tuples satisfying two conditions: (1) the transaction date lies in the given year and (2) the customer associated to the sales fact was unmarried on the transaction date. Therefore, to check both of these conditions, at least two accesses to the time dimension table are required. Longer codes are required if the time involves a time range other than a particular year, such as from the second quarter of 1999 to the first quarter of 2000. Code to access and check the time condition is of a repetitive type, which is not desirable. Moreover, if the query involves more than one slowly changing attribute, similar examinations need to be performed repeatedly.

Considering this kind of business questions, which is common in reality, a modification of the basic schema is proposed in the next section.

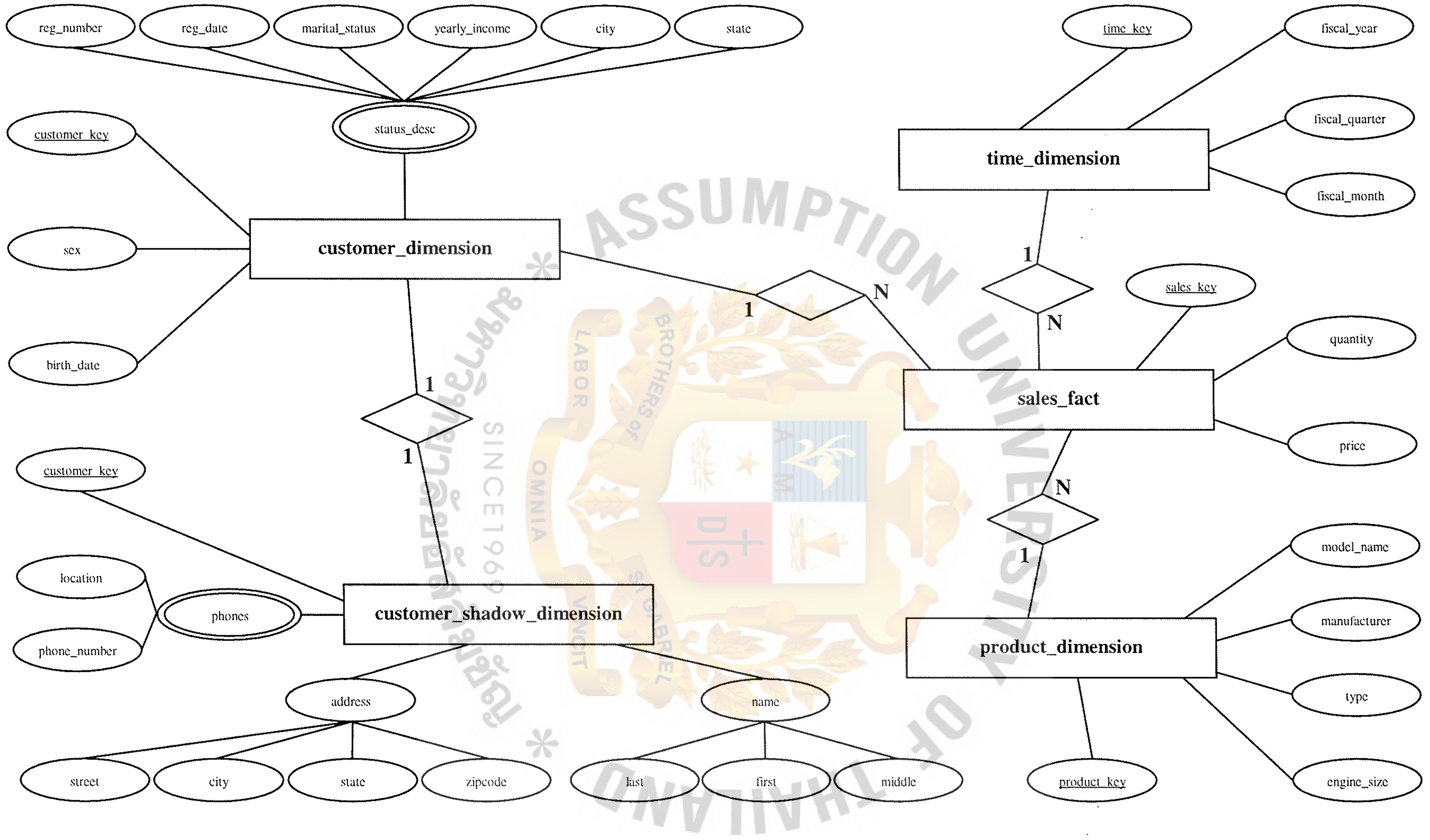

Figure 4.3. Modified ER Diagram of Warehouse Database.

4.3 Modification of the Warehousing Database Schema

Although the schema for the warehousing database described in the last section, which effectively makes use of the time to deal with slowly changing dimension attributes, seems to be theoretically sound, answering time-related queries becomes a little complicated. Some modification of the schema is made to handle time-related queries more efficiently.

The modification is based on the assumption that the customer registers on the same day of purchase. And therefore, the values of slowly changing attributes of "customer_dimension" table are the values on the date of registration, and the exact date from when the attributes actually changed. If the customer marries on July 15, 1999 prior to buying a car on August 15, 1999, the "from_date_key" captures the later date. This is unavoidable, because the transaction database just records the date of registration. Further, a car dealer database is not supposed to keep track of the customer's status change.

If a row of slowly changing attributes (which are multivalued composite type) are added at each purchase indicating the attribute value at purchase time, irrespective of whether the attribute is changed or not, query formulation could be easier. For a time-related query, facts can be first filtered out based on time (using the attributes of "time_dimension" table), followed by a simple matching of attribute value and corresponding date. In fact, the attribute "to_time_key" is found to be redundant in this case and therefore, can be deleted. Therefore, for each slowly changing attribute, the attribute value at the time of each purchase is recorded, even if the value remains unchanged.

This solution leads to identical "from_time_key" for each changing attribute of the "customer_dimension" table. The redundant "from_time_key" attributes can be eliminated, by having only one multivalued composite attribute "status" instead of four. Accordingly, the ER Diagram of the data warehouse is modified and represented in Figure 4.3. Only the attributes of the "customer_dimension" table have been modified. The rest of the diagram is same as Figure 4.2. Implementation of this schema is described in the next chapter.

V. IMPLEMENTATION OF THE DATA WAREHOUSE

In the last Chapter, the schema of an object-relational data warehouse was developed. The schema has been implemented using an ORDBMS. This chapter is devoted towards the implementation process.

The Oracle 8i (version 8.1.6) database server by Oracle Corporation was chosen to implement the schema. Although this version of Oracle server was released in 1999, it has all the basic features of object-relational technology. In 2001, Oracle Corporation released Oracle 9i, but there was no major change as far as object-relational features are concerned.

The reasons behind choosing Oracle among the leading ORDBMS are twofold. The primary one is that the author had access to one of the Oracle Servers, which makes this work possible. The other is the author's familiarity with Oracle database servers and their associated documentation. The schema developed in Chapter 4 was DBMS independent, implying that it could have been implemented using any DBMS product supporting object-relational technology.

This chapter starts with the Data Definition Language (DDL) statements in Section 5.1 to build the objects and tables associated with the warehouse database. Section 5.2 shows the Data Manipulation Languages (DML) to perform insert, update, and delete operations. Finally in Section 5.3, queries are presented to support business questions. Oracle's standard SQL3 language is used to formulate the SQL statements. To improve readability, uppercase letters are used for SQL keywords.

5.1 Building Objects and Tables

As discussed before, the implemented database schema is based on Figure 4.3 of Chapter 4. For easy understandings of the mapping of entities and attributes of the ER Diagram to tables, some entities and attributes of the original ER Diagram were renamed. The modified ER Diagram, presented in Figure 5.1, gives an overview of all the tables and the relationships between them as they are implemented.

The section starts with the development of relatively simple time dimension and product dimension tables. This is followed by relatively more complex ones like the customer shadow dimension and customer dimension tables. Finally, the sales fact table is described.

5.1.1 Time Dimension Table

The time dimension table, time_dimension_objtab, consists of simple attributes. The attribute "time_key" is the primary key of the table. Attributes "fiscal_year", "fiscal_quarter", and "fiscal_month" signify the associated value to which a time key belongs. For example, a "time_key" 2002Q3M09 belongs to fiscal year 2002, fiscal quarter 3, and fiscal month September.

As the rows of time_dimension_objtab table are to be referenced by other tables (like "sales_fact_objtab", using "REF" data-type), to maintain primary key / foreign key equivalent relationship in the RDBMS, they need to be declared as objects. This could be done by creating an object-type "time_dimension_objtyp" as shown in the following statement:

```
CREATE TYPE time_dimension_objtyp AS OBJECT (
   time_key CHAR(10),
   fiscal_year NUMBER(4),
   fiscal_quarter CHAR(10),
   fiscal_month CHAR(10));
```

The data-types of the attributes are defined inside the object-type definition. This newly created object-type can now be used to define any new object or table. The constraints such as primary key, index, etc., are the attributes of a table and hence, shall be specified during table definition.

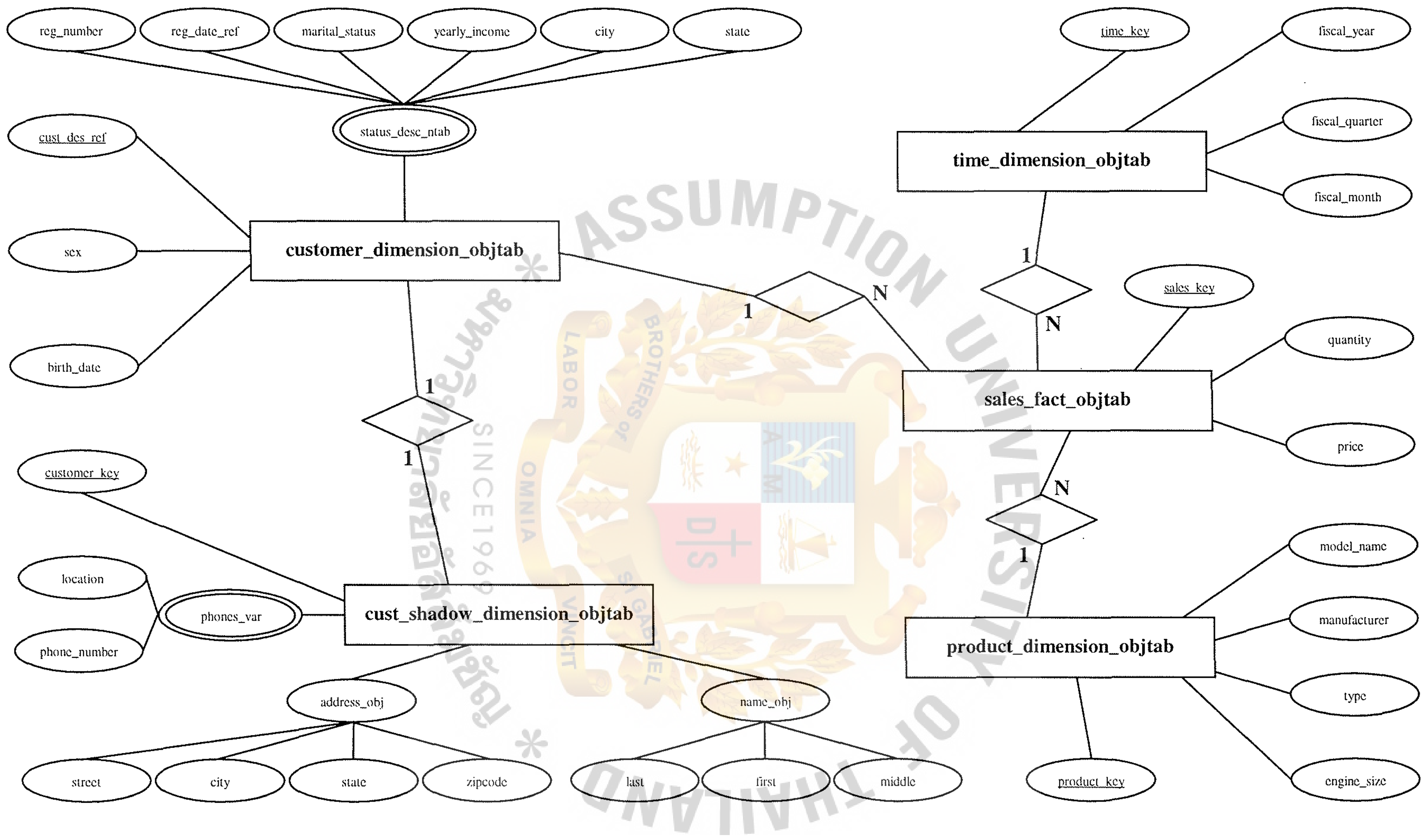

Figure 5.1. ER Diagram of Implemented Warehouse Database.

The time_dimension_objtyp table is created next using the following statement:

```
CREATE TABLE time_dimension_objtab OF time_dimension_objtyp (
   PRIMARY KEY (time_key),
   fiscal_year NOT NULL,
   fiscal_quarter NOT NULL,
   fiscal_month NOT NULL)
   OBJECT ID SYSTEM GENERATED;
```

The rows of the newly created table are objects of "time_dimension_objtyp". As this table is made of objects, it can also be called an object table. The last part of the table name signifies so. The row objects of the table can be referenced from other objects. This is demonstrated later.

Each row object of the table has an Object Identifier (ID), by which the row object can be referenced. This Object ID (OID) could be a 16 byte system generated globally unique identification, or else it could be the primary key of the table, which is locally unique. This is specified in the last line, "OBJECT ID SYSTEM GENERATED", signifying that the OID for this table is a system generated 16 byte ID. This is the default. The primary key can also be made OID by specifying "OBJECT ID PRIMARY KEY". The choice of primary key as OID may be more efficient in cases where the primary key value is smaller than the default 16 byte system generated identifier (Russell 1999). An object-relational representation of the time dimension table is shown in Figure 5.2.

Table time_dimension_objtab (of time_dimension_objtyp)

time_key	fiscal_year	fiscal_quarter	fiscal_month
Text CHAR(10)	Nember NUMBER(4)	Text CHAR(10)	Text CHAR(10)
PK	NOT NULL	NOT NULL	NOT NULL

Figure 5.2. Object-Relational Representation of the Time Dimension Table.

The constraints on the table (primary key and not null) are specified during the table definition.

5.1.2 Product Dimension Table

Like the time dimension table, the product dimension table, product_dimension_objtab, also consists of simple attributes. The primary key of this table is "product_key". Other attributes are "model_name", "manufacturer", "type", and "engine_size". An object-type "product_dimension_objtyp" is first created followed by the table definition. The SQL statements are identical to those for the time dimension table described in section 5.1.1. An object-relational representation of the product dimension table is shown in Figure 5.3. Following SQL statements are used to build the product_dimension_objtab table.

```
CREATE TYPE product_dimension_objtyp AS OBJECT (
   product_key CHAR(6),
   model_name VARCHAR2(30),
   manufacturer VARCHAR2(25),
   type VARCHAR2(50),
   engine_size CHAR(6));

CREATE TABLE product_dimension_objtab OF product_dimension_objtyp(
   PRIMARY KEY (product_key),
   model_name NOT NULL,
   manufacturer NOT NULL,
   type NOT NULL,
   engine_size NOT NULL)
   OBJECT ID SYSTEM GENERATED;
```

Table product_dimension_objtab (of product_dimension_objtyp)

product_key	model_name	manufacturer	type	engine_size
Text CHAR(6)	Text VARCHAR2(30)	Text VARCHAR2(25)	Text VARCHAR2(50)	Text CHAR(6)
PK	NOT NULL	NOT NULL	NOT NULL	NOT NULL

Figure 5.3. Object-Relational Representation of the Product Dimension Table.

5.1.3 Customer Shadow Dimension Table

The customer shadow dimension table, cust_shadow_dimension_objtab, consists of simple, composite, and multivalued composite attributes. Figure 5.4 shows an object-relational representation of the complete structure of cust_shadow_dimension_objtab table.

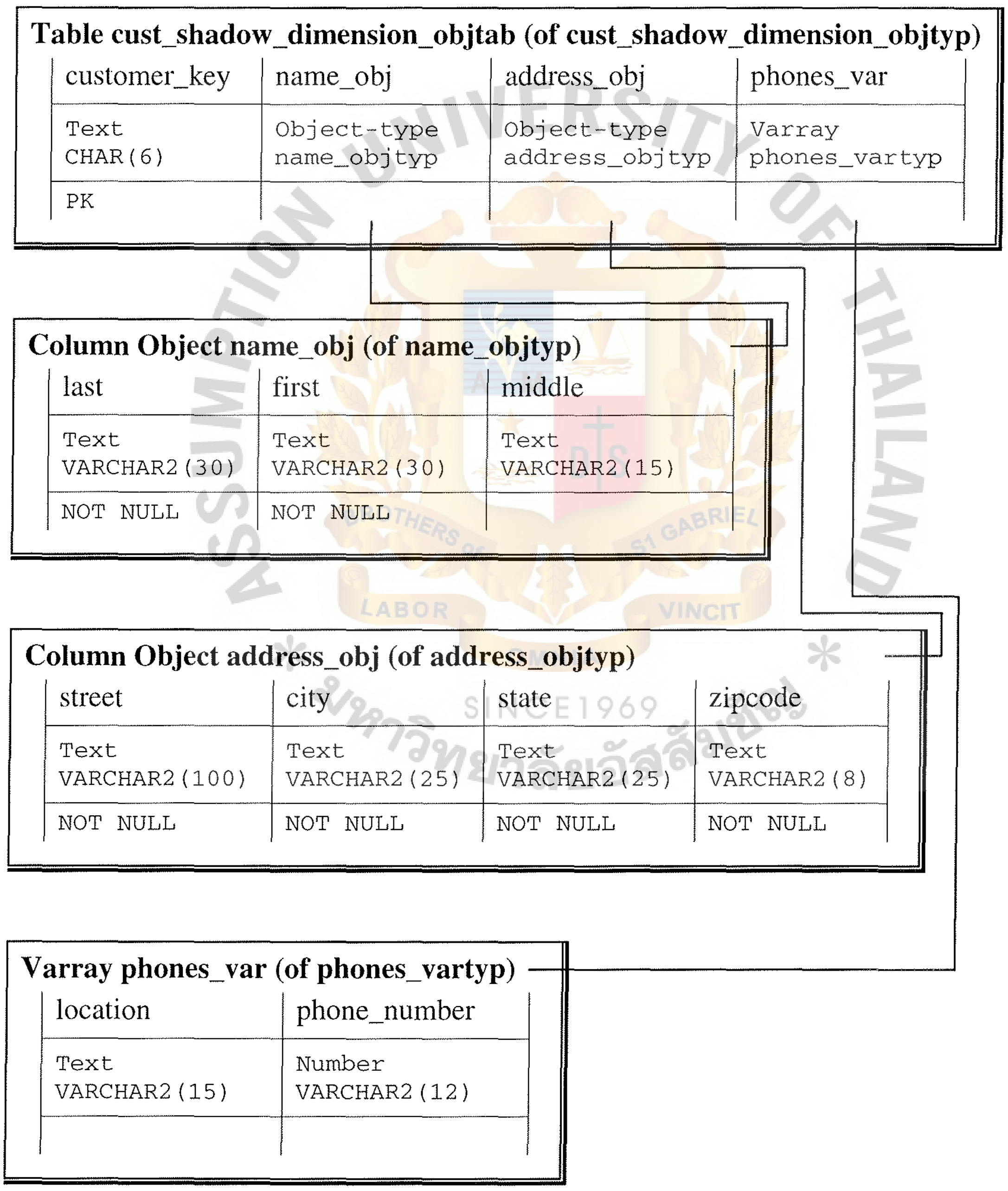

Table cust_shadow_dimension_objtab (of cust_shadow_dimension_objtyp)

customer_key	name_obj	address_obj	phones_var
Text CHAR(6)	Object-type name_objtyp	Object-type address_objtyp	Varray phones_vartyp
PK			

Column Object name_obj (of name_objtyp)

last	first	middle
Text VARCHAR2(30)	Text VARCHAR2(30)	Text VARCHAR2(15)
NOT NULL	NOT NULL	

Column Object address_obj (of address_objtyp)

street	city	state	zipcode
Text VARCHAR2(100)	Text VARCHAR2(25)	Text VARCHAR2(25)	Text VARCHAR2(8)
NOT NULL	NOT NULL	NOT NULL	NOT NULL

Varray phones_var (of phones_vartyp)

location	phone_number
Text VARCHAR2(15)	Number VARCHAR2(12)

Figure 5.4. Object-Relational Representation of the Customer Shadow Dimension Table.

Oracle considers a multivalued composite attribute (e.g. phones_var) as a collection of composite attributes. Two collection data-types are supported – "varray" and nested table. For "varray", the order of elements is defined. The maximum number of elements is also predefined in the case of "varray", although the upper bound can be changed. A nested table can have any number of elements and behaves as a regular table where update, delete, and insert operations are possible. The order of elements is not defined in the case of a nested table. If the entire collection is to be manipulated as a single set, "varray" is a better option. Update is not supported for "varray" (Russell 1999).

In case of "phones_var" of cust_shadow_dimension_objtab table, phone numbers are to be treated as a single set. It is assumed that no data manipulation will be done based on "phones_var" Therefore "phones_var" is treated as data-type "varray".

For both multivalued composite and composite attributes, the underlying composite attribute is defined first as a new object-type. This is done using the following statements:

```
CREATE TYPE name_objtyp AS OBJECT (
   last VARCHAR2(30),
   first VARCHAR2(30),
   middle VARCHAR2(15));

CREATE TYPE address_objtyp AS OBJECT (
   street VARCHAR2(100),
   city VARCHAR2(25),
   state VARCHAR2(25),
   zipcode VARCHAR2(8));

CREATE TYPE phones_objtyp AS OBJECT (
   location VARCHAR2(15),
   phone_number VARCHAR2(12));
```

The following statement defines the "varray" data-type phones_vartyp. In this case the "varray" is designed to hold a maximum of 10 elements of phones_objtyp object-type.

```
CREATE TYPE phones_vartyp AS VARRAY(10) OF phones_objtyp;
```

Once the building blocks are created, cust_shadow_dimension_objtyp is defined. Finally the object table cust_shadow_dimension_objtab is created, each row of which is a cust_shadow_dimension_objtyp object-type. The associated SQL statements are presented below:

```
CREATE TYPE cust_shadow_dimension_objtyp AS OBJECT (
   customer_key CHAR(6),
   name_obj name_objtyp,
   address_obj address_objtyp,
   phones_var phones_vartyp);

CREATE TABLE cust_shadow_dimension_objtab OF
   cust_shadow_dimension_objtyp (
   PRIMARY KEY (customer_key),
   CHECK (name_obj.last IS NOT NULL),
   CHECK (name_obj.first IS NOT NULL),
   CHECK (address_obj.street IS NOT NULL),
   CHECK (address_obj.city IS NOT NULL),
   CHECK (address_obj.state IS NOT NULL),
   CHECK (address_obj.zipcode IS NOT NULL))
   OBJECT ID SYSTEM GENERATED;
```

5.1.4 Customer Dimension Table

The customer dimension table, customer_dimension_objtab, is probably the most complex table in this study. The structure of customer_dimension_objtab is shown in Figure 5.5. It contains a multivalued complex attribute "status_desc" which is used to deal with slowly changing attributes of the dimension table. This collection is a nested table, as the individual elements of the collection are to be accessed and the collection needs to be updated each time an old customer purchases a new car. The statements to create a nested table are similar to "varray". The composite is defined using a new object-type, and a table of that object-type is created. The following statements created the nested table:

```
CREATE TYPE status_desc_objtyp AS OBJECT (
   reg_number CHAR(3),
   reg_date_ref REF time_dimension_objtyp,
   marital_status VARCHAR2(10),
   yearly_income NUMBER,
   city VARCHAR2(25),
   state VARCHAR2(25));

CREATE TYPE status_desc_ntabtyp AS TABLE OF status_desc_objtyp;
```

Once the nested table has been created, the row object-type of customer_dimension_objtab is defined. To do so, a new object-type called customer_dimension_objtyp is created. The definition of customer_dimension_objtyp contains a reference (REF) data-type "cust_des_ref". "REF" is a built-in data-type of ORACLE. It is a logical pointer, which points to a row object of another object table.

As shown in Figure 5.5, "cust_des_ref" is created to point to a row object of "cust_shadow_dimension_objtyp" object-type.

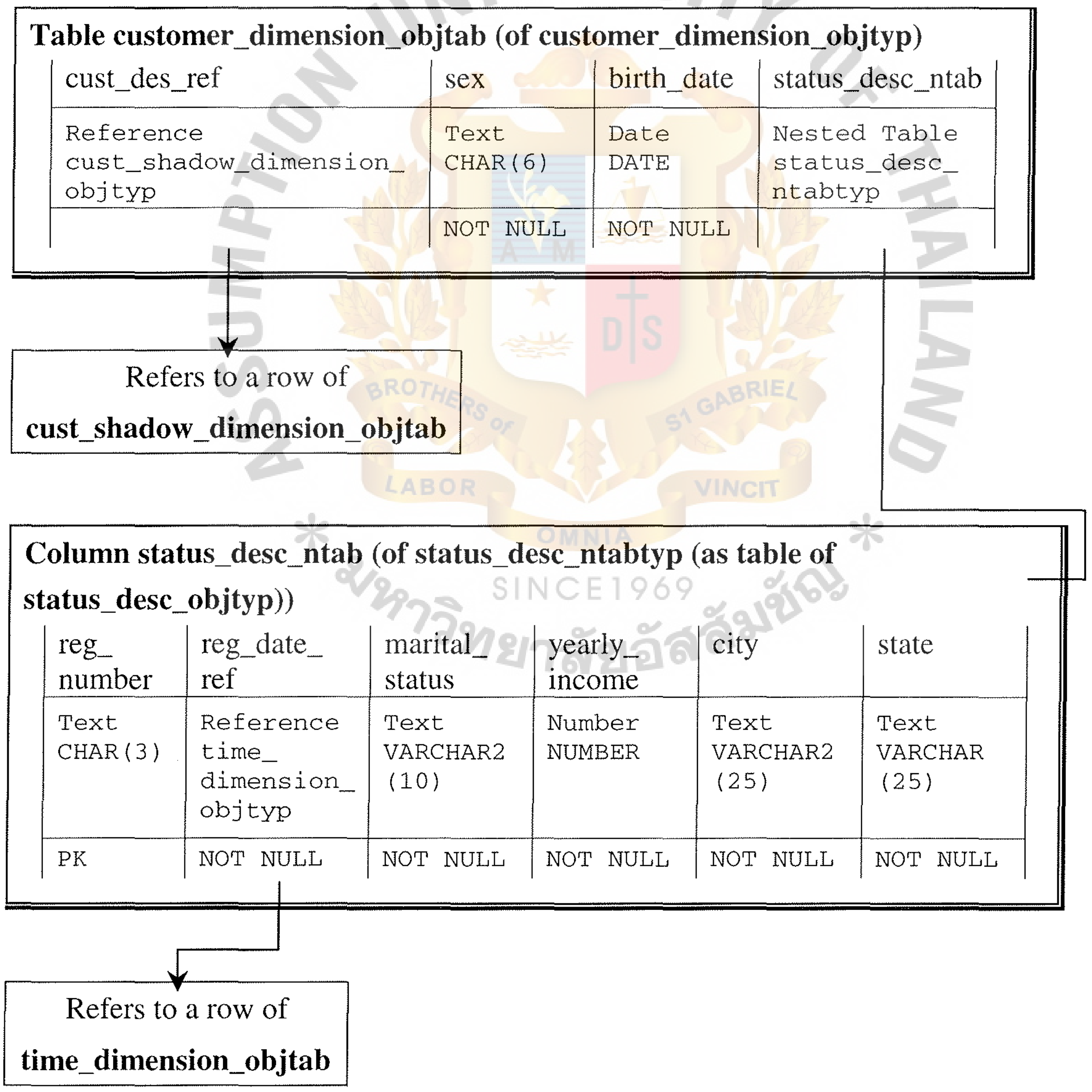

Table customer_dimension_objtab (of customer_dimension_objtyp)

cust_des_ref	sex	birth_date	status_desc_ntab
Reference cust_shadow_dimension_objtyp	Text CHAR(6)	Date DATE	Nested Table status_desc_ntabtyp
	NOT NULL	NOT NULL	

Refers to a row of
cust_shadow_dimension_objtab

Column status_desc_ntab (of status_desc_ntabtyp (as table of status_desc_objtyp))

reg_number	reg_date_ref	marital_status	yearly_income	city	state
Text CHAR(3)	Reference time_dimension_objtyp	Text VARCHAR2 (10)	Number NUMBER	Text VARCHAR2 (25)	Text VARCHAR (25)
PK	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL

Refers to a row of
time_dimension_objtab

Figure 5.5. Object-Relational Representation of the Customer Dimension Table.

"REF" allows association among the objects. It reduces the need for foreign keys. It also provides an easy mechanism for navigating between objects. The "cust_des_ref" joins the customer_dimension_objtab and cust_shadow_dimension_objtab tables, which have a one to one relationship among them. It is to be noted that the object composite data-type for nested table "status_desc_objtyp" also has a "REF" data-type, "reg_date_ref", which helps to establish the one-to-many relationship between the nested table of customer_dimension_objtab and type_dimension_objtab.

Object-type "customer_dimension_objtyp" is created as follows:

```
CREATE TYPE customer_dimension_objtyp AS OBJECT (
   cust_des_ref REF cust_shadow_dimension_objtyp,
   sex CHAR(6),
   birth_date DATE,
   status_desc_ntab status_desc_ntabtyp);
```

Finally, the table "customer_dimension_objtab" is created using the following statement:

```
CREATE TABLE customer_dimension_objtab OF
   customer_dimension_objtyp (
   FOREIGN KEY (cust_des_ref) REFERENCES
     cust_shadow_dimension_objtab,
   cust_des_ref NOT NULL,
   sex NOT NULL,
   birth_date NOT NULL)
   OBJECT ID SYSTEM GENERATED
   NESTED TABLE status_desc_ntab STORE AS status_desc_store_ntab (
      (PRIMARY KEY (NESTED_TABLE_ID, reg_number),
      reg_date_ref NOT NULL)
      ORGANIZATION INDEX COMPRESS);
```

There is no primary key for the table defined above. In fact, "cusr_des_ref" pointer, which is a foreign key relating this table and the cust_shadow_dimension_table with a one-to-one relationship, is the primary key of the table. However, surprisingly, Oracle does not allow the "REF" data-type to be a primary key or part of a primary key. As each row of the customer_dimension_objtab object table is an object and each object can have unique system generated OID, there is no exclusive need of having a primary key for this table.

The nested table is stored separately. A table name is to be given for the nested table, using which it will be stored, although the table is not directly accessible using that name. All the constraints of the nested table are also defined here. However the scope of any "REF" data-type of the nested table could not be defined within the main table statement. The scope of "reg_date_ref" is defined separately as follows:

```
ALTER TABLE status_desc_store_ntab
   ADD (SCOPE FOR (reg_date_ref) IS time_dimension_objtab);
```

5.1.5 Sales Fact Table

The central fact table, sales_fact_objtab, has a relatively simple structure, which is described in Figure 5.6.

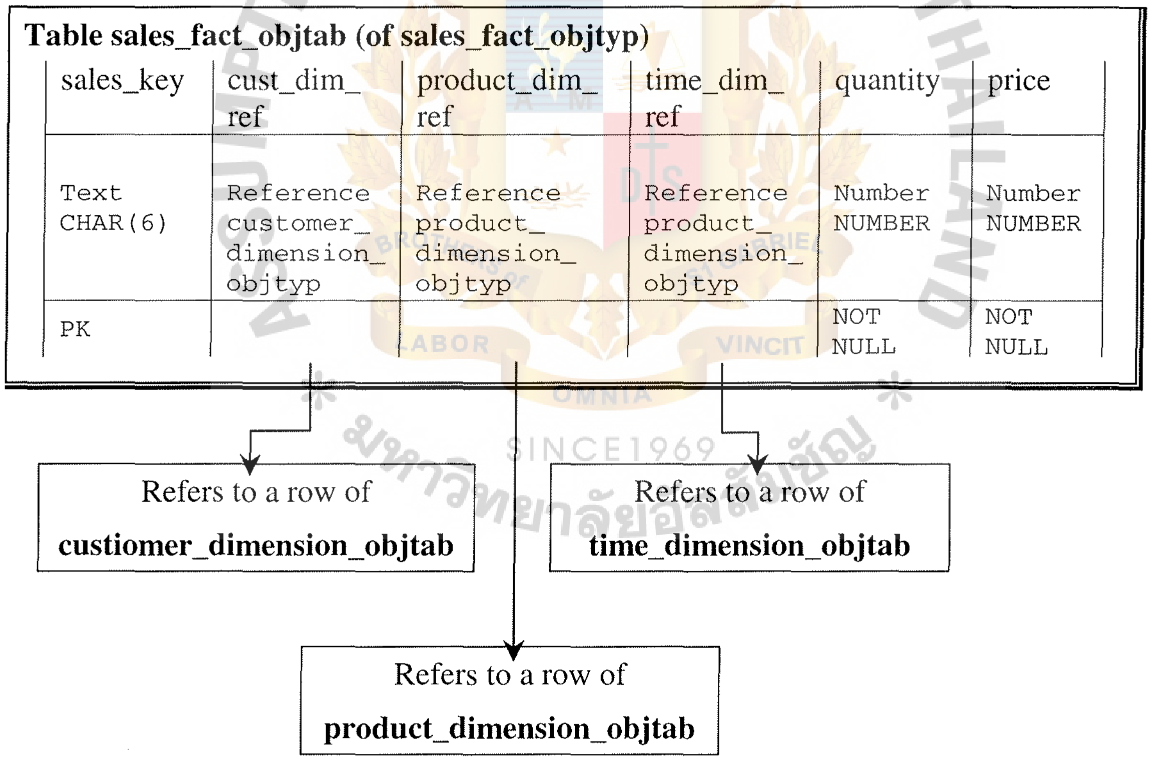

Table sales_fact_objtab (of sales_fact_objtyp)

sales_key	cust_dim_ ref	product_dim_ ref	time_dim_ ref	quantity	price
Text CHAR(6)	Reference customer_ dimension_ objtyp	Reference product_ dimension_ objtyp	Reference product_ dimension_ objtyp	Number NUMBER	Number NUMBER
PK				NOT NULL	NOT NULL

Refers to a row of **custiomer_dimension_objtab**
Refers to a row of **time_dimension_objtab**
Refers to a row of **product_dimension_objtab**

Figure 5.6. Object-Relational Representation of the Sales Fact Table.

Like other tables, a row object-type "sales_fact_objtyp" is created first. It has three "REF" data-type pointers pointing to the data-types of the dimension tables. Based on this new object-type, the object table sales_fact_objtab is created. The DDL statements for creation of sales_fact objtyp and sales_fact_objtab are provided below:

```
CREATE TYPE sales_fact_objtyp AS OBJECT (
   sales_key CHAR(6),
   cust_dim_ref REF customer_dimension_objtyp,
   prod_dim_ref REF product_dimension_objtyp,
   time_dim_ref REF time_dimension_objtyp,
   quantity number,
   price number);

CREATE TABLE sales_fact_objtab OF sales_fact_objtyp (
   PRIMARY KEY (sales_key),
   FOREIGN KEY (cust_dim_ref) REFERENCES
      customer_dimension_objtab,
   FOREIGN KEY (prod_dim_ref) REFERENCES product_dimension_objtab,
   FOREIGN KEY (time_dim_ref) REFERENCES time_dimension_objtab,
   cust_dim_ref NOT NULL,
   prod_dim_ref NOT NULL,
   time_dim_ref NOT NULL,
   quantity NOT NULL,
   price NOT NULL)
   OBJECT ID SYSTEM GENERATED;
```

The foreign key constraints define the scope of the RFF data-types naturally. Other constraints of the sales_fact_objtab table are incorporated in the table definition.

5.2 Data Manipulation – Insert, Update, and Delete Operations

Conventionally, data warehouses contain read-only data. Data correction is not required frequently. However, update and delete is required to cope up with changing attributes of the dimension. If an attribute of a table changes with time, either the old data needs to be replaced with the new data, or only the new data is inserted leaving the old data untouched. Deletion is a rare possibility, particularly when data storage is inexpensive. Irrespective of the above, in this section, all three basic data handling functions – insert, update, and delete will be examined.

Like the last section, this section is divided into five sub-sections, each of which deals with one of the object tables.

5.2.1 Time Dimension Table

A typical data insertion statement in the time dimension table, time_dimension_objtab, looks as follows:

```
INSERT INTO time_dimension_objtab VALUES (
   '1999Q1M01', 1999, 'Quarter 1', 'January');
```

Data update could be made using a statement similar to the following:

```
UPDATE time_dimension_objtab
   SET fiscal_year = 1998,
   fiscal_quarter = 'Quarter 2',
   fiscal_month = 'April'
   WHERE time_key = '1999Q1M01';
```

Deletion could be performed using the following statement:

```
DELETE FROM time_dimension_objtab
   WHERE time_key = '1999Q1M01';
```

In this study, the primary key of the time dimension table, i.e. the "time_key", was assigned meaningful values. This was done intentionally with the sole purpose of easy understanding. The advantages of meaningful values were not applied during query formulation. In practice, meaningful primary keys should generally be avoided, as it may lead to the necessity of re-valuing or re-sequencing the primary key values in the future.

5.2.2 Product Dimension Table

Insert, update, and delete statements on the product dimension table, product_dimension_objtab, are as simple as on the time dimension table described above. The insert statement looks like the following:

```
INSERT INTO product_dimension_objtab VALUES (
   'P001',
   'Corolla',
   'Toyota',
   'CE 4dr Sedan (4cyl, 4A)',
   '1800CC');
```

Data could be updated with the following statement:

```
UPDATE product_dimension_objtab
   SET model_name = 'Civic',
   manufacturer = 'General Motors',
   type = '2 Door Sedan',
   engine_size = '500CC'
   WHERE product_key = 'P001';
```

The following statement is for deletion:

```
DELETE FROM product_dimension_objtab
   WHERE product_key = 'P001';
```

As in a relational database, a row cannot be deleted if another table, using a scoped "REF" statement, references the row object.

5.2.3 Customer Shadow Dimension Table

Until this point the DML statements were similar to those in a relational database, as the associated tables were simple. Complexity arises with composite data structures, particularly if they are collection types, e.g. "varray" or nested tables. A typical insert statement for the customer shadow dimension table, cust_shadow_dimension_objtyp, is as follows:

```
INSERT INTO cust_shadow_dimension_objtab VALUES (
   'C001',
   name_objtyp ('Abbott', 'Jonathan', NULL),
   address_objtyp ('1421 S Sheridan Road', 'Tulsa', 'Oklahoma',
      '74112'),
   phones_vartyp (
      phones_objtyp ('Home', '918-835-3161'),
      phones_objtyp ('Mobile', '338-331-4463')));
```

The following statement shows a complete update of a customer's record, when an old customer registers again to purchase a new car.

```
UPDATE cust_shadow_dimension_objtab
   SET name_obj = name_objtyp ('Clinton', 'Bill', NULL),
   address_obj = address_objtyp ('2922/282 New Petchburi road',
      'Westside', 'New York', '00000'),
   phones_var = phones_vartyp (
      phones_objtyp ('Home', '001-000-0000'),
      phones_objtyp ('Office', '002-999-9999'),
      phones_objtyp ('Mobile', '003-111-1111'))
   WHERE customer_key = 'C001';
```

In case the customer has no contact number, the following statement sets the phones_var value to NULL:

```
UPDATE cust_shadow_dimension_objtab
   SET name_obj = name_objtyp ('Clinton', 'Bill', NULL),
   address_obj = address_objtyp ('2922/282 New Petchburi road',
      'Westside', 'new York', '00000'),
   phones_var = NULL
   WHERE customer_key = 'C001';
```

Deletion of a record could be done with the following statement:

```
DELETE FROM cust_shadow_dimension_objtab
   WHERE customer_key = 'C001';
```

5.2.4 Customer Dimension Table

Since it is the most complicated of all the tables in the database, data manipulation is complex for customer dimension table, customer_dimension_objtab. The situation becomes worse when a new row is inserted in the nested table without changing the attributes of the main table. This is considered as an insert from the nested table's point of view, but the main table sees it as an update. Furthermore, this kind of operation is required every time an old customer buys a new car. The main table also has attributes of "REF" data-types "cust_ref_des" and "reg_date_ref", which point to rows of the customer shadow dimension and time dimension tables respectively. To insert this "REF" data-type, select statements on parent tables are required. As discussed before, the "REF" data-type enforces the data integrity establishing an equivalent relationship parallel to a primary key / foreign key in a relational database.

When a customer purchases a car for the first time, an insert operation is required similar to the following:

```
INSERT INTO customer_dimension_objtab VALUES (
   (SELECT REF(csdim) FROM cust_shadow_dimension_objtab csdim
      WHERE customer_key = 'C001'),
   'Male',
   '22-DEC-1964',
   status_desc_ntabtyp (
      status_desc_objtyp ('R01',
      (SELECT REF(t) FROM time_dimension_objtab t WHERE
         time_key = '1999Q3M09'),
      'Unmarried', 60000, 'Tulsa', 'Oklahoma')));
```

After a couple of years, if the same customer purchases another new car, a new row will be inserted in the nested table alone using the following statement:

```
INSERT INTO TABLE (
   SELECT cdim.status_desc_ntab
   FROM customer_dimension_objtab cdim
   WHERE cdim.cust_des_ref =
      (SELECT REF(csdim) FROM cust_shadow_dimension_objtab csdim
         WHERE customer_key = 'C001'))
   VALUES ('R02',
      (SELECT REF(t) FROM time_dimension_objtab t WHERE
         time_key = '2002Q1M03'),
      'Unmarried', 70000, 'Skokie', 'Illinois');
```

Like insert, update statements for main and nested tables are handled differently. Update of main table is straightforward and is shown below:

```
UPDATE customer_dimension_objtab
   SET sex = 'Female',
   birth_date = '20-MAY-1960'
   WHERE cust_des_ref =
      (SELECT REF(csdim) FROM cust_shadow_dimension_objtab csdim
         WHERE customer_key = 'C001');
```

Piecewise update of "varray" is not possible, the entire array needs to be rewritten. For nested table, however, any part can be updated as shown below:

```
UPDATE TABLE (
   SELECT cdim.status_desc_ntab
   FROM customer_dimension_objtab cdim
   WHERE cdim.cust_des_ref =
      (SELECT REF(csdim) FROM cust_shadow_dimension_objtab csdim
         WHERE customer_key = 'C001'))
   SET reg_date_ref = (SELECT REF(t) FROM time_dimension_objtab t
      WHERE time_key = '2002Q2M06'),
   yearly_income = '100000'
   WHERE reg_number = 'R02';
```

Deletion of an entire row of the main table is done as follows:

```
DELETE FROM customer_dimension_objtab
   WHERE cust_des_ref =
      (SELECT REF(csdim) FROM cust_shadow_dimension_objtab csdim
         WHERE customer_key = 'C001');
```

To delete a row from the nested table, the following statement can be used:

```
DELETE TABLE (
   SELECT cdim.status_desc_ntab
   FROM customer_dimension_objtab cdim
   WHERE cdim.cust_des_ref =
      (SELECT REF(csdim) FROM cust_shadow_dimension_objtab csdim
         WHERE customer_key = 'C001'))
   WHERE reg_number = 'R02';
```

5.2.5 Sales Fact Table

The attributes of sales fact table, sales_fact_objtab, are simple type. This fact table is linked with the dimension tables using "REF" data-type pointing to row objects of the dimension object tables. New rows can be inserted into the sales_fact_objtab table using the following statement:

```
INSERT INTO sales_fact_objtab VALUES (
   'S0001',
   (SELECT REF(cdim) FROM customer_dimension_objtab cdim
      WHERE cust_des_ref = (
      SELECT REF(csdim) FROM cust_shadow_dimension_objtab csdim
      WHERE customer_key = 'C001')),
   (SELECT REF(p) FROM product_dimension_objtab p
      WHERE product_key = 'P004'),
   (SELECT REF(t) FROM time_dimension_objtab t
      WHERE time_key = '1999Q3M09'),
   1,
   21550);
```

Rows can be updated using the following statement:

```
UPDATE sales_fact_objtab
   SET cust_dim_ref = (SELECT REF(cdim) FROM
      customer_dimension_objtab cdim WHERE cust_des_ref = (
      SELECT REF(csdim) FROM cust_shadow_dimension_objtab csdim
      WHERE customer_key = 'C002')),
   prod_dim_ref = (SELECT REF(p) FROM product_dimension_objtab p
      WHERE product_key = 'P006'),
   time_dim_ref = (SELECT REF(t) FROM time_dimension_objtab t
      WHERE time_key = '1999Q3M09')
   WHERE sales_key = 'S0001';
```

Deletion of a record is also simple. There could be two possible situations for deletion. The following statement deletes a record having a specific "sales_key":

```
DELETE sales_fact_objtab
   WHERE sales_key = 'S0001';
```

If a record belonging to a specific customer is to be deleted, the following command could be issued:

```
DELETE sales_fact_objtab
   WHERE cust_dim_ref = (SELECT REF(cdim) FROM
      customer_dimension_objtab cdim WHERE cust_des_ref = (
         SELECT REF(csdim) FROM cust_shadow_dimension_objtab csdim
      WHERE customer_key = 'C001'));
```

5.3 Query Formulation

The objective of the entire exercise, from database design to implementation using object-relational technology, is to obtain correct query results and ease of query formulation. While calculating results, the database should be able to handle time-related queries involving slowly changing attributes of the dimension tables. Ease of query formulation is essential for ad-hoc queries, which is essential for executive support systems.

This section will examine a few queries addressing common business questions. For easy understanding, simple queries involving less complexity will be discussed first, followed by harder ones.

A simple but common inquiry is to determine number of cars sold in a certain year, for example during fiscal year 2000. This question involves the fact table and time dimension table, which are bound together using the "REF" data-type attribute "time_dim_ref". Unlike queries on relational tables, for object-relational database, no exclusive join statement is required, which makes the query simple. The ORDBMS makes an internal join between the two tables, which is more efficient. The query looks like the following:

```
SELECT count(*)
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000;
```

The time dimension table is accessed from the sales fact table through "REF" data-type. These tables, together with the relationship between them, logically behave like a single table.

The next query serves to answer how many cars manufactured by Jeep were sold during fiscal year 2000. This involves two dimension tables, time and product, in addition to the fact table. The query is formulated below:

```
SELECT count(*)
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.prod_dim_ref.manufacturer = 'Jeep';
```

This was similar to the last one. An additional statement was included to satisfy the manufacturer constraint.

The next example involves a sub-query. It finds which manufacturer had the maximum sales amount (i.e. total price) during fiscal year 2000. The statement looks like the following:

```
SELECT sf1.prod_dim_ref.manufacturer
FROM sales_fact_objtab sf1
WHERE sf1.time_dim_ref.fiscal_year = 2000
GROUP BY sf1.prod_dim_ref.manufacturer
HAVING SUM(sf1.price) >= ALL (SELECT SUM(sf2.price) FROM
   sales_fact_objtab sf2
   WHERE sf2.time_dim_ref.fiscal_year = 2000 GROUP BY
   sf2.prod_dim_ref.manufacturer);
```

Object-relational technology eliminates the need of a join statement, but the sub-query is unavoidable.

The next query serves to answer how many cars manufactured by Jeep were sold to male customers during fiscal year 2000. Clearly this involves all three dimension tables – customer, product, and time. As all three tables are bounded by “REF” data-type with the fact table, formulation of the query is similar to the ones discussed earlier as is shown below:

```
SELECT count(*)
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.prod_dim_ref.manufacturer = 'Jeep'
AND sf.cust_dim_ref.sex = 'Male';
```

The next query demonstrates the use of the built-in functions of the standard SQL language on “REF” data-type. Suppose it is required to find out how many male customers purchased a car during fiscal year 2000. The query looks like the following:

```
SELECT COUNT(DISTINCT sf.cust_dim_ref)
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.cust_dim_ref.sex = 'Male';
```

The above query is similar to the ones in relational SQL language.

Until this point, the queries have not involved the nested table of the customer dimension database. The nested table contains the status attributes, which are slowly changing in reality, and the nested table captures the value of the attributes at the time of the purchase of each car. Fortunately, Oracle allows handling the nested table as a normal relational one. The only difference is that the nested table has to be declared in the FROM clause. If it is required to identify the number of different (distinct) divorced female customers from Massachusetts who purchased cars manufactured by Jeep during fiscal year 2000, the following query could be used:

```
SELECT COUNT(DISTINCT sf.cust_dim_ref)
FROM sales_fact_objtab sf,
   TABLE (sf.cust_dim_ref.status_desc_ntab) sdes
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.cust_dim_ref.sex = 'Female'
AND sdes.marital_status = 'Divorced'
AND sdes.state = 'Massachusetts'
AND sdes.reg_date_ref = sf.time_dim_ref;
```

In the above statement, the nested table is identified by the alias "sdes" in the FROM clause. Subsequently the alias "sdes" is used like an ordinary relational table name. The sales fact records are filtered out for the required time period (in this case, fiscal year 2000) only once, using the "REF" data-type of fact table pointing to the time dimension table. Later (in the last line), when the validity of the attribute value (in this case, state name) is to be checked against time, the time reference of the attribute is simply equated with the previously filtered time key. This is how this database structure eliminates the need for rechecking time for each slowly changing attribute to be considered in the query. This was the main reason for modifying the original warehouse database. This saves a massive amount of repetitive code when formulating queries.

Another example, similar to the last one, involving two slowly changing attributes of the customer table would be to find the total quantity and price amount of cars having engine size 4000cc or more, sold to unmarried customers with a yearly income of 75,000 or less during first quarter of the fiscal year 2000.

```
SELECT SUM(sf.quantity), SUM(sf.price)
FROM sales_fact_objtab sf,
   TABLE (sf.cust_dim_ref.status_desc_ntab) sdes
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.time_dim_ref.fiscal_quarter = 'Quarter 1'
AND sf.prod_dim_ref.engine_size >= '4000CC'
AND sdes.marital_status = 'Unmarried'
AND sdes.yearly_income <= 75000
AND sdes.reg_date_ref = sf.time_dim_ref;
```

The following query identifies a single customer from the customer shadow dimension table, which is connected to the sales fact table indirectly using two "REF" data-types, "cust_dim_ref" and "cust_des_ref". The query also handles a complex time-related inquiry. Suppose it is required to find out which customer purchased the maximum number of cars from quarter 3 of fiscal year 1999 to quarter 2 of fiscal year 2000. This can be solved using the following query, which also includes a sub-query to resolve the "maximum" issue.

```
SELECT sf1.cust_dim_ref.cust_des_ref.customer_key,
   sum(sf1.quantity)
FROM sales_fact_objtab sf1
WHERE ((sf1.time_dim_ref.fiscal_year = 1999
      AND sf1.time_dim_ref.fiscal_quarter = 'Quarter 3')
   OR (sf1.time_dim_ref.fiscal_year = 1999
      AND sf1.time_dim_ref.fiscal_quarter = 'Quarter 4')
   OR (sf1.time_dim_ref.fiscal_year = 2000
      AND sf1.time_dim_ref.fiscal_quarter = 'Quarter 1')
   OR (sf1.time_dim_ref.fiscal_year = 2000
      AND sf1.time_dim_ref.fiscal_quarter = 'Quarter 2'))
GROUP BY sf1.cust_dim_ref.cust_des_ref.customer_key
HAVING SUM(sf1.quantity) >= ALL (SELECT SUM(sf2.quantity)
   FROM sales_fact_objtab sf2
   WHERE ((sf2.time_dim_ref.fiscal_year = 1999
         AND sf2.time_dim_ref.fiscal_quarter = 'Quarter 3')
      OR (sf2.time_dim_ref.fiscal_year = 1999
         AND sf2.time_dim_ref.fiscal_quarter = 'Quarter 4')
      OR (sf2.time_dim_ref.fiscal_year = 2000
         AND sf2.time_dim_ref.fiscal_quarter = 'Quarter 1')
      OR (sf2.time_dim_ref.fiscal_year = 2000
         AND sf2.time_dim_ref.fiscal_quarter = 'Quarter 2'))
   GROUP BY sf2.cust_dim_ref.cust_des_ref.customer_key);
```

The customer shadow dimension table assists in identifying a customer's most recent information such as name, address, or phone number. The following query reveals the details of customers who purchased cars manufactured by BMW during the fiscal year 2000.

```
SELECT sf.cust_dim_ref.cust_des_ref.name_obj,
   sf.cust_dim_ref.cust_des_ref.address_obj,
   sf.cust_dim_ref.cust_des_ref.phones_var
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.prod_dim_ref.manufacturer = 'BMW';
```

Pure time-related problems could be answered with ease. A common question would be to identify the quarter of a year which generally experienced the most sales. The solution to the above follows:

```
SELECT sf1.time_dim_ref.fiscal_quarter
FROM sales_fact_objtab sf1
GROUP BY sf1.time_dim_ref.fiscal_quarter
HAVING SUM(sf1.quantity) >= ALL (SELECT SUM(sf2.quantity)
   FROM sales_fact_objtab sf2
   GROUP BY sf2.time_dim_ref.fiscal_quarter);
```

The last query of this section is to rank customers by marital status according to the number of cars purchased during the fiscal year 2000. This type of problem is common in data warehousing. This warehouse database can handle it efficiently as follows:

```
SELECT sdes.marital_status, SUM(sf.quantity)
FROM sales_fact_objtab sf,
   TABLE (sf.cust_dim_ref.status_desc_ntab) sdes
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sdes.reg_date_ref = sf.time_dim_ref
GROUP BY sdes.marital_status
ORDER BY 2;
```

VI. RESULTS AND DISCUSSION

The objective of this project was to study the effect of object-relational technology on data warehousing. It was to examine whether object-relational technology can be used to build an improved data warehouse, which is commonly built with a relational database. It was also to explore whether the object-relational database can handle the slowly changing dimension attributes in a better way than a relational database.

The study was conducted in two phases. In the first phase, which is described in Chapter IV, a database was developed, based on an operational database, to fulfill the warehousing needs. Although the warehouse database was theoretically acceptable, some problems were encountered while formulating queries. This led to a modification of the database – all the five nested tables handling slowly changing dimensions were combined together to form one, which was expected to improve the ease of querying. In the second phase, implementation of the database was demonstrated in Chapter V using Oracle 8i.

Two important improvements were achieved while using object-relational technology in data warehousing – efficient handling of slowly changing dimension attributes and ease of query formulation. These are discussed in detail in Sections 6.1 and 6.2 respectively. Some of the technical difficulties of object-relational databases are discussed in Section 6.3.

6.1 Dealing with Slowly Changing Attributes

Ideally, a data warehouse is nonvolatile in nature and contains only read-only data. New tuples are appended frequently in the fact table, which is usual. Occasionally, tuples may be appended in the dimension table as well, for example in the case study, if a new customer purchases a car. Although update of existing data in the dimension table does not happen theoretically, it is not uncommon in real life. In this case study, the annual income of the existing customer could be changed during the second or subsequent times of purchase. If slowly changing attributes are not handled correctly, it may lead to incorrect analysis.

As described in Section II, different solutions were suggested to handle slowly changing attributes, all of them based on relational technology. Giovinazzo 2000 summarized different approaches as listed below:

(1) Record Overwriting:

A new value overwrites an old dimension attribute value. This wipes out the old data permanently destroying the integrity of the data warehouse and therefore may produce an incorrect analysis.

(2) New Record Creation:

A new record containing updated attributes value is appended, without making any change to old record. This solution cannot relate the old record with the new one. In this case study, this may result in losing valuable information about the customer's behavior.

(3) Old/Current Fields:

The structure of dimension record is changed to accommodate both the old and new values of the dimension record. The solution is not realistic as it is difficult to predict beforehand the number of times the attribute values may be changed.

(4) Record Versioning:

A version number field is provided in addition to the primary key of the dimension record to keep track of the change. Although this solution is theoretically correct, it complicates the relationship between the fact and dimension tables. Preparation of a query becomes a difficult issue.

(5) Record Linking:

When the dimension changes, a new record is created with a new primary key. The client ID is carried over from the operational system and is used to link different versions of dimension records to one another. According to Giovinazzo 2000 this is the best approach discovered so far using relational database.

This study makes use of object-relational technology to deal with slowly changing dimensions. This is accomplished using two techniques. First, the values of changing attributes are stored in the record itself using a nested table, without inserting any additional row as the case was for a relational database. Second, a time reference field is added, similar to a temporal database, to determine the period during which the attribute value was valid. Both of them together provide the database with unique capabilities to handle slowly changing attributes. The strengths of this approach over any of the five techniques described above using relational database are the following:

(1) All values of changing attributes are stored inside the entity's record, which improves logical thinking about the entity. Each record describes the entity completely.

(2) As one entity is represented by exactly one record, it is simple to navigate the dimension table, which can be traced directly from the fact table.

(3) As attributes of an entity are stored exactly once, less space is required for storage.

(4) For the same reason as mentioned in Point 3, insert, update, and delete operations on an entity record is easier and faster.

Contrary to the common belief that the complex data structure of this model may complicate the formulation of query, object-relational technology actually simplifies query formulation. This is described in the next section.

6.2 Formulation of Queries

Data definition (i.e. construction of the tables) and data manipulation (i.e. insert, update, and delete operations) on an object-relational warehouse database are discussed in Section 5.1 and Section 5.2 of Chapter V. If compared with a relational database, the statements seem to be more complex. This becomes prominent when multivalued (e.g. nested table and "varray") and composite objects are encountered.

However, creation of objects, tables, and views, is a one-time event. Once those are created, there is no need to touch them again unless the database structure needs to be modified, which is rare in the case of a good design. Data manipulation (i.e. mapping from a transaction processing database to a warehouse database) is generally done periodically using automated software where the associated statements are defined once. Therefore complication of data definition and data manipulation (insert, update, and delete) statements may not be a drawback

The objective of data warehousing is to satisfy analytical needs. Therefore ease of "select" query formulation is probably one of the most important requirements of a data warehouse. Besides form-based queries, ad-hoc queries on data warehouse are common.

Section 5.3 of Chapter V described how to handle queries related to business questions using the object-relational database. Despite the presence of nested tables and complex objects, formulation of queries was found to be shorter and easier when compared to a relational database. The reasons are the following:

(1) Using the "REF" data-type to build primary key and foreign key relationships eliminates the need of code exclusively for joining the tables.

Tables are joined internally, which make the navigation easier. This can be considered a major advantage. In a real world data warehouse, there may be many dimension tables, and joining them together in each query requires a lot of code and effort. The dot (.) operator of object-relational SQL allows joining of tables quickly whenever required.

(2) Querying nested tables, which are designed to handle slowly changing dimensions in this study, is similar to querying ordinary tables. The only small difference is that the nested table is defined by an alias in the "FROM" clause. Therefore little extra effort is required in querying nested tables.

6.3 Drawbacks of Object-Relational Database

Although an object-relational database was found to be advantageous over a relational database in building a data warehouse, it should not be considered perfect. One of the major weaknesses of this technology is lack of standardization. Although it is there in the commercial database market for the last few years, the user interface language, SQL3 is not yet consistent. All leading vendors use their own version of the language. Although they are conceptually the same, users need to refer to the vendors' specifications to work with the database.

The second problem, which is worthwhile to mention here, is also due to non-standardization of object-relational database technology. Different vendors treat multivalued differently in their products. Oracle supports "varray" and nested tables, each of which has its own strengths. Some other commercial DBMS call them set, bag, or list. Those may lead to confusion if the specific DBMS manual is not referred to. The method of creation of objects and table structure and manipulation of data (insert, update, and delete) differs significantly among different DBMS.

The above problems would be resolved in the near future after standardization of object-relational database is achieved. However, the strengths of the technology probably outweigh the drawbacks, which make it suitable, at the very least, for data warehousing needs.

VII. CONCLUSIONS AND RECOMMENDATIONS

Being a hybrid of relational database and object database, the object-relational model certainly enjoys an edge over both. This study has utilized some of the features of object-relational technology to build a warehouse data model, which was implemented using Oracle 8i DBMS. In this chapter, the key findings from this study are summarized and those are presented in Section 7.1. Section 7.2 states some recommendations for future work to strengthen the conclusions arrived at in this study.

7.1 Conclusions

This study concludes that object-relational technology assists in building an improved data warehouse, one that is better than those based on conventional technologies, including relational database, with respect to the following considerations:

(1) Object-relational allows multivalued attributes (e.g., in case of Oracle 8i, nested tables). Multivalued attributes assist in representing entities with slowly changing attributes effectively. This ensures correct data analysis in an efficient way.

(2) The formulation of queries, which need frequent joins of the fact table with multiple dimension tables, is easier and needs fewer lines of code if an object-relational database is used.

However, this study is unable to arrive at a conclusion on performance issues, as those depend on the individual implementation of the database such as the hardware platform, operating system, physical access mechanism, and query optimizer.

7.2 Recommendations

There is scope for further research in this field. This study has been carried out at a conceptual level involving a limited number of tables and records of a sample database. This is probably not enough to arrive at a definitive conclusion. More studies with practical data need to be carried out to support the conclusions derived from this study. The cost of development of a data warehouse using object-relational technology also needs to be compared with that of a warehouse built on conventional technology.

The implementation of the data warehouse was done at a very basic level, using SQL statements. However, leading DBMS vendors provide software tools to map from transaction database to warehouse database. Due to the unavailability of any commercially available product, the data model developed here could not be verified against such software tools. Doing so, however, is essential in order to evaluate the practical usability of the model developed in this report.

APPENDIX A

TEST DATA

Table A.1. Test Data of the time_dimension_objtab Table.

time_key (CHAR(10))	fiscal_year (NUMBER(4))	fiscal_quarter (CHAR(10))	fiscal_month (CHAR(10))
1999Q1M01	1999	Quarter 1	January
1999Q1M02	1999	Quarter 1	February
1999Q1M03	1999	Quarter 1	March
1999Q1M04	1999	Quarter 2	April
1999Q1M05	1999	Quarter 2	May
1999Q1M06	1999	Quarter 2	June
1999Q1M07	1999	Quarter 3	July
1999Q1M08	1999	Quarter 3	August
1999Q1M09	1999	Quarter 3	September
1999Q1M10	1999	Quarter 4	October
1999Q1M11	1999	Quarter 4	November
1999Q1M12	1999	Quarter 4	December
2000Q1M01	2000	Quarter 1	January
2000Q1M02	2000	Quarter 1	February
2000Q1M03	2000	Quarter 1	March
2000Q1M04	2000	Quarter 2	April
2000Q1M05	2000	Quarter 2	May
2000Q1M06	2000	Quarter 2	June
2000Q1M07	2000	Quarter 3	July
2000Q1M08	2000	Quarter 3	August
2000Q1M09	2000	Quarter 3	September
2000Q1M10	2000	Quarter 4	October
2000Q1M11	2000	Quarter 4	November
2000Q1M12	2000	Quarter 4	December
2001Q1M01	2001	Quarter 1	January
2001Q1M02	2001	Quarter 1	February
2001Q1M03	2001	Quarter 1	March
2001Q1M04	2001	Quarter 2	April
2001Q1M05	2001	Quarter 2	May
2001Q1M06	2001	Quarter 2	June
2001Q1M07	2001	Quarter 3	July
2001Q1M08	2001	Quarter 3	August
2001Q1M09	2001	Quarter 3	September
2001Q1M10	2001	Quarter 4	October
2001Q1M11	2001	Quarter 4	November
2001Q1M12	2001	Quarter 4	December
2002Q1M01	2002	Quarter 1	January
2002Q1M02	2002	Quarter 1	February
2002Q1M03	2002	Quarter 1	March
2002Q1M04	2002	Quarter 2	April
2002Q1M05	2002	Quarter 2	May
2002Q1M06	2002	Quarter 2	June

Table A.2. Test Data of the product_dimension_objtab Table.

product_key (CHAR(6))	model_name (VARCHAR2(30))	manufacturer (VARCHAR2(25))	type (VARCHAR2(50))	engine_size (CHAR(6))
P001	Corolla	Toyota	CE 4dr Sedan (4cyl, 4A)	1800CC
P002	RAV4	Toyota	4WD 4dr SUV (4cyl, 4A)	2000CC
P003	Accord	Honda	EX 4dr Sedan (4cyl, 4A)	2300CC
P004	CR-V	Honda	EX AWD 4dr SUV (4cyl, 4A)	2400CC
P005	Golf	Volkswagen	GLS TDI 4dr Hatchback (4cyl, 4A)	1900CC
P006	BMW 5 Series	BMW	540i 4dr Sedan (8cyl, 5A)	4400CC
P007	A4 Series	Audi	Quattro AWD 4dr Sedan (6cyl, 6A)	3000CC
P008	Crown Victoria	Ford	LX 4dr Sedan (8cyl, 4A)	4600CC
P009	Grand Cherokee	Jeep	Laredo 4WD 4dr SUV (8cyl, 5A)	4700CC
P010	GS 300	Lexus	4dr Sedan (6cyl, 5A)	3000CC

Table A.3. Test Data of the cust_shadow_dimension_objtab Table.

customer_key	name_obj			address_obj				phones_var	
	last	first	middle	street	city	state	zipcode	location	phone_number
(CHAR(6))	(VARCHAR2(30))	(VARCHAR2(30))	(VARCHAR2(15))	(VARCHAR2(100))	(VARCHAR2(25))	(VARCHAR2(25))	(VARCHAR2(25))	(VARCHAR2(15))	(VARCHAR2(12))
C001	Abbott	Jonathan		1421 S Sheridan Road	Tulsa	Oklahoma	74112	Home	918-835-3161
								Mobile	338-331-4463
C002	Alexandar	Stephen	P	1700 West Loop South	Houston	Texas	77027	Home	713-621-9720
								Office	713-565-7865
								Mobile	834-736-6935
C003	Allen	Norene		15 Southwest Park	Westwood	Massachusetts	02090	Home	781-329-3350
								Office	781-329-9875
								Mobile	456-453-9899
C004	Loffredo	Robert		9922 Roosevelt Boulveard	Edwardsville	Pennsylvania	18704	Home	570-283-1860
								Office	215-676-0989
								Mobile	610-921-0719
C005	Shefali	Mehta		3424 Peachtree Road	Boston	Massachusetts	02116	Home	617-572-2022
								Office	782-453-8963
C006	Chimenya	Esther		588 North Gulph Road	New York	New York	11373	Home	712-525-3636
								Office	712-527-4242
								Mobile	263-783-9898
C007	Cheesbrough	John		128 Intervale Road	Burlington	Texas	75001	Home	972-661-8960
								Office	972-381-0817
								Mobile	445-677-6555
C008	Holmquist	Jim	Alberto	11711 Berryessa Road	San Jose	California	23434	Home	408-487-3138
								Office	408-288-4081
								Mobile	737-784-7474
C009	Claydon	Graham		11220 Allisonville Road	Fishers	Indiana	64038	Home	317-842-1040
								Office	381-784-8432
								Mobile	633-734-7843
C010	Conway	Warwick		5801 E. 76th Avenue	Commerce City	Colorado	80022	Home	303-286-0406
								Office	303-637-9192
								Mobile	722-373-9643

Table A.4. Test Data of the customer_dimension_objtab Table.

cust_des_ref	sex	birth_date	status_desc_ntab					
			reg_number	reg_date_ref	marital_status	yearly_income	city	state
(REF Datatype)	(CHAR(6))	(DATE)	(CHAR(3))	(REF Datatype)	VARCHAR2(10)	(NUMBER)	(VARCHAR2(25))	(VARCHAR2(25))
C001	Male	22-12-1964	R01	1999Q3M09	Unmarried	60000	Tulsa	Oklahoma
			R02	2001Q1M01	Married	66000	Skokie	Illinois
			R03	2002Q1M03	Married	70000	Tulsa	Oklahoma
C002	Male	03-04-1973	R01	1999Q4M11	Unmarried	100000	Nashua	New Hampshire
			R02	2000Q2M06	Unmarried	100000	Houston	Texas
C003	Female	14-06-1956	R01	1999Q4M12	Married	90000	Westwood	Massachusetts
			R02	2000Q3M09	Divorced	120000	Westwood	Massachusetts
			R03	2001Q4M12	Married	150000	Westwood	Massachusetts
C004	Male	21-05-1967	R01	1999Q4M12	Married	50000	Reading	Pennsylvania
			R02	2001Q4M12	Married	60000	Hawthorne	New Jersey
			R03	2002Q2M06	Married	70000	Edwardsville	Pennsylvania
C005	Female	06-02-1956	R01	2000Q1M03	Married	100000	Chicago	Illinois
			R02	2002Q1M03	Married	100000	Boston	Massachusetts
C006	Female	14-04-1972	R01	2000Q1M03	Unmarried	45000	King of Prussia	Philadelphia
			R02	2001Q2M06	Married	20000	Ottawa	Illinois
			R03	2002Q2M05	Divorced	60000	New York	New York
C007	Male	08-09-1964	R01	2000Q2M06	Married	90000	Burlington	Texas
C008	Male	12-10-1953	R01	2000Q3M08	Married	100000	Redwood City	California
			R02	2002Q1M01	Married	120000	San Jose	California
C009	Female	15-09-1971	R01	2000Q1M03	Unmarried	65000	Elizabethtown	Kentucky
			R02	2002Q2M05	Married	70000	Fishers	Indiana
C010	Male	24-08-1969	R01	2000Q3M08	Unmarried	75000	Las Vegas	Nevada
			R02	2000Q4M11	Married	95000	Salt Lake City	Utah
			R03	2001Q4M12	Divorced	110000	Commerce City	Colorado

Table A.5. Test Data of the sales_fact_objtab Table.

sales_key (CHAR(6))	cust_dim_ref (REF Datatype)	prod_dim_ref (REF Datatype)	time_dim_ref (REF Datatype)	quantity (NUMBER)	price (NUMBER)
S0001	C001	P004	1999Q3M09	1	21,550
S0002	C001	P006	2001Q1M01	1	48,650
S0003	C001	P001	2002Q1M03	1	15,550
S0004	C002	P002	1999Q4M11	1	18,750
S0005	C002	P008	2000Q2M06	1	16,500
S0006	C003	P001	1999Q4M12	1	14,330
S0007	C003	P005	2000Q3M09	1	17,725
S0008	C003	P010	2001Q4M12	1	37,650
S0009	C004	P003	1999Q4M12	1	22,440
S0010	C004	P007	2001Q4M12	1	31,340
S0011	C004	P004	2002Q2M06	1	22,705
S0012	C005	P009	2000Q1M03	1	28,670
S0013	C005	P008	2002Q1M03	1	27,780
S0014	C006	P006	2000Q1M03	1	50,885
S0015	C006	P005	2001Q2M06	1	18,560
S0016	C006	P007	2002Q2M05	1	31,800
S0017	C007	P004	2000Q2M06	1	23,655
S0018	C008	P003	2000Q3M08	1	22,760
S0019	C008	P001	2002Q1M01	1	14,655
S0020	C009	P006	2000Q1M03	1	51,845
S0021	C009	P008	2002Q2M05	1	27,785
S0022	C010	P005	2000Q3M08	1	18,650
S0023	C010	P009	2000Q4M11	1	29,605
S0024	C010	P010	2001Q4M12	1	39,180

APPENDIX B

SAMPLE SQL QUERY AND RESULT

1. How many cars were sold during the fiscal year 2000?

Query:

```
SELECT count(*) AS NO_OF_CAR
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000;
```

Result:

```
NO_OF_CAR
----------
         9
```

2. How many cars manufactured by Jeep were sold during the fiscal year 2000?

Query:

```
SELECT count(*) AS NO_OF_CAR_OF_JEEP_MANUFACTURER
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.prod_dim_ref.manufacturer = 'Jeep';
```

Result:

```
NO_OF_CAR_OF_JEEP_MANUFACTURER
------------------------------
                             2
```

3. Which manufacturer had the maximum sales amount (i.e. total price) during the fiscal year 2000?

Query:

```
SELECT sf1.prod_dim_ref.manufacturer AS MANUFACTURER
FROM sales_fact_objtab sf1
WHERE sf1.time_dim_ref.fiscal_year = 2000
GROUP BY sf1.prod_dim_ref.manufacturer
HAVING SUM(sf1.price) >= ALL (SELECT SUM(sf2.price)
   FROM sales_fact_objtab sf2
   WHERE sf2.time_dim_ref.fiscal_year = 2000
   GROUP BY sf2.prod_dim_ref.manufacturer);
```

Result:

```
MANUFACTURER
-------------------------
BMW
```

4. How many cars manufactured by Jeep were sold to male customers during the fiscal year 2000?

Query:

```
SELECT count(*) AS NO_OF_CAR_OF_JEEP_MANUFACTURER
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.prod_dim_ref.manufacturer = 'Jeep';
```

Result:

```
NO_OF_CAR_OF_JEEP_MANUFACTURER
------------------------------
                             1
```

5. How many married male customers purchased a car during the fiscal year 2000?

Query:

```
SELECT COUNT(DISTINCT sf.cust_dim_ref) AS
   MARRIED_MALE_CUSTOMER
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.cust_dim_ref.sex = 'Male';
```

Result:

```
NO_OF_MARRIED_MALE_CUSTOMER
---------------------------
                          4
```

6. How many different (distinct) divorced female customers from Massachusetts purchased car manufactured by Jeep during the fiscal year 2000?

Query:

```
SELECT COUNT(DISTINCT sf.cust_dim_ref) AS
   NO_OF_DIVORCED_FEMALE_CUSTOMER
FROM sales_fact_objtab sf,
   TABLE (sf.cust_dim_ref.status_desc_ntab) sdes
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.cust_dim_ref.sex = 'Female'
AND sdes.marital_status = 'Divorced'
AND sdes.state = 'Massachusetts'
AND sdes.reg_date_ref = sf.time_dim_ref;
```

Result:

```
NO_OF_DIVORCED_FEMALE_CUSTOMER
------------------------------
                             1
```

7. What was the total price amount and quantity of cars having engine size 4000CC or more sold to unmarried customer having yearly income $75,000 or less during the first quarter of fiscal year 2000?

Query:

```
SELECT SUM(sf.price) AS SALES_AMOUNT, SUM(sf.quantity) AS
   SALES_QUANTITY
FROM sales_fact_objtab sf,
   TABLE (sf.cust_dim_ref.status_desc_ntab) sdes
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.time_dim_ref.fiscal_quarter = 'Quarter 1'
AND sf.prod_dim_ref.engine_size >= '4000CC'
AND sdes.marital_status = 'Unmarried'
AND sdes.yearly_income <= 75000
AND sdes.reg_date_ref = sf.time_dim_ref;
```

Result:

```
SALES_AMOUNT SALES_QUANTITY
------------ --------------
      102730              2
```

8. What is the customer_key and number of cars purchased for customer(s) who purchased maximum cars from fiscal quarter 3 of fiscal year 1999 to fiscal quarter 2 of fiscal year 2000?

Query:

```
SELECT sf1.cust_dim_ref.cust_des_ref.customer_key AS
   CUSTOMER_KEY, sum(sf1.quantity) AS NO_OF_CAR
FROM sales_fact_objtab sf1
WHERE ((sf1.time_dim_ref.fiscal_year = 1999 AND
   sf1.time_dim_ref.fiscal_quarter = 'Quarter 3')
OR (sf1.time_dim_ref.fiscal_year = 1999 AND
   sf1.time_dim_ref.fiscal_quarter = 'Quarter 4')
OR (sf1.time_dim_ref.fiscal_year = 2000 AND
   sf1.time_dim_ref.fiscal_quarter = 'Quarter 1')
OR (sf1.time_dim_ref.fiscal_year = 2000
   AND sf1.time_dim_ref.fiscal_quarter = 'Quarter 2'))
```

```
GROUP BY sf1.cust_dim_ref.cust_des_ref.customer_key
HAVING SUM(sf1.quantity) >= ALL (SELECT SUM(sf2.quantity)
   FROM sales_fact_objtab sf2
   WHERE ((sf2.time_dim_ref.fiscal_year = 1999 AND
      sf2.time_dim_ref.fiscal_quarter = 'Quarter 3')
   OR (sf2.time_dim_ref.fiscal_year = 1999 AND
      sf2.time_dim_ref.fiscal_quarter = 'Quarter 4')
   OR (sf2.time_dim_ref.fiscal_year = 2000 AND
      sf2.time_dim_ref.fiscal_quarter = 'Quarter 1')
   OR (sf2.time_dim_ref.fiscal_year = 2000 AND
      sf2.time_dim_ref.fiscal_quarter = 'Quarter 2'))
   GROUP BY sf2.cust_dim_ref.cust_des_ref.customer_key);
```

Result:

```
CUSTOMER_KEY  NO_OF_CAR
------------ ----------
C002                 2
```

9. Which customers purchased car manufactured by BMW during the fiscal year 2000?

Query:

```
SELECT sf.cust_dim_ref.cust_des_ref.name_obj AS
   CUSTOMER_NAME, sf.cust_dim_ref.cust_des_ref.address_obj
   AS CUSTOMER_ADDRESS,
   sf.cust_dim_ref.cust_des_ref.phones_var AS
   CUSTOMER_PHONE
FROM sales_fact_objtab sf
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sf.prod_dim_ref.manufacturer = 'BMW';
```

Result:

```
CUSTOMER_NAME(LAST, FIRST, MIDDLE)
-----------------------------------------------------------
CUSTOMER_ADDRESS(STREET, CITY, STATE, ZIPCODE)
-----------------------------------------------------------
CUSTOMER_PHONE(LOCATION, PHONE_NUMBER)
-----------------------------------------------------------
NAME_OBJTYP('Chimenya', 'Esther', NULL)
ADDRESS_OBJTYP('588 North Gulph Road', 'New York',
   'New York', '11373')
PHONES_VARTYP(PHONES_OBJTYP('Home', '712-525-3636'),
   PHONES_OBJTYP('Office', '712-527-4242'),
   PHONES_OBJTYP('Mobile', '263-783-9898'))
```

```
NAME_OBJTYP('Claydon', 'Graham', NULL)
ADDRESS_OBJTYP('11220 Allisonville Road', 'Fishers',
   'Indiana', '64038')
PHONES_VARTYP(PHONES_OBJTYP('Home', '317-842-1040'),
   PHONES_OBJTYP('Office', '381-784-8432'),
   PHONES_OBJTYP('Mobile', '633-734-7843'))
```

10. Based on past trend, which quarter of the year generally experiences more car sales?

Query:

```
SELECT sf1.time_dim_ref.fiscal_quarter AS QUARTER_NAME
FROM sales_fact_objtab sf1
GROUP BY sf1.time_dim_ref.fiscal_quarter
HAVING SUM(sf1.quantity) >= ALL (SELECT SUM(sf2.quantity)
   FROM sales_fact_objtab sf2
   GROUP BY sf2.time_dim_ref.fiscal_quarter);
```

Result:

```
QUARTER_NAME
------------
Quarter 1
Quarter 4
```

11. Order customers by marital status according to the number of car purchased during the fiscal year 2000.

Query:

```
SELECT sdes.marital_status AS MARITAL_STATUS,
   SUM(sf.quantity) AS NO_OF_CAR
FROM sales_fact_objtab sf,
   TABLE (sf.cust_dim_ref.status_desc_ntab) sdes
WHERE sf.time_dim_ref.fiscal_year = 2000
AND sdes.reg_date_ref = sf.time_dim_ref
GROUP BY sdes.marital_status
ORDER BY 2;
```

Result:

```
MARITAL_STATUS  NO_OF_CAR
-------------- ----------
Divorced                1
Unmarried               4
Married                 4
```

BIBLIOGRAPHY

1. Agrawal, Rakesh, Ashish Gupta, and Sunita Sarawagi. "Modeling Multidimensional Databases," Proceedings of the 13th International Conference on Data Engineering – 1997, Institute of Electrical and Electronics Engineers, Birmingham, 1997.

2. Armstrong, Rob. "Seven Steps to Optimizing Data Warehouse Performance." IEEE Computer 34, no. 12 (December, 2001): 76-79.

3. Anahory, S. and D. Murray. Data Warehousing in the Real World. England: Addison-Wesley, 1997.

4. Bahrami, A. Object-Oriented Systems Development. Illinious: Irwin/McGraw-Hill, 1998.

5. Chaudhuri, S. and U. Dayal. "An Overview of Data Warehousing and OLTP Technology." ACM Sigmod Record 26, no. 1 (1997).

6. Chelluri, Kiran and Vijay Kumar. "Data Classification and Management in Very Large Data Warehouses," Proceedings of the Third International Workshop on Advances Issues of E-Commerce and Web-Based Information System (WECWIS'01), Institute of Electrical and Electronics Engineers, San Juan, California, June 21-22, 2001.

7. Chen, Wei-Chou, Tzung-Pei Hong, and Wei-Yang Lin. "A Composite Data Model in Object-Oriented Data Warehousing," Proceedings of the 31st International Conference on Technology of Object-Oriented Language and Systems, Institute of Electrical and Electronics Engineers, Nanjing, China, September 22-25, 1999.

8. Date, C. J. An Introduction to Database Systems. Massachusetts: Addison-Wesley, 1999.

9. Elmasri, R. A. and S. B. Navathe. Fundamentals of Database Systems. California: Addison-Wesley, 2000.

10. Furlow, Gerri. "The Case for Building a Data Warehouse." IEEE IT Professional 3, no. 4 (July/August, 2001): 31-34.

11. Giovinazzo, W.A. Object-Oriented Data Warehouse Design: A Star Schema. New Jersey: Prentice Hall PTR, 2000.

12. Hanson, Joseph H. and Mary Jane Willshire. "Modeling a Faster Data Warehouse," Proceedings of the International Database Engineering and Applications Symposium (IDEAS'97), Institute of Electrical and Electronics Engineers, Montreal, Canada, August 24-27, 1997.

13. Inmon, W. H. Building the Data Warehouse. Wellesley: John Wiley & Sons, 2002.

14. Inmon, W. H. and C. Kelley. Rdb/VMS: Developing the Data Warehouse. Boston: QED Publishing Group, 1993.

15. Kimball, R. "A Dimensional Modeling Manifesto." DBMS Online 10, no. 9 (1997).

16. Kimball, R. The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses. New York: J. Wiley, 1996.

17. Russell, J. Oracle 8i: Application Developer's Guide - Object-Relational Features, Release 2 (8.1.6). Redwood City: Oracle Corporation, 1999.

18. Sarda, N. L. "Temporal Issues in Data Warehouse Systems," Proceedings of the International Symposium on Database Applications in Non-Traditional Environments (DANTE'99), Institute of Electrical and Electronics Engineers, Koyto, Japan, November 28-30, 1999.

19. Singh, H. Data Warehousing. New Jersey: Prentice Hall PTR, 1998.

20. Stonebraker, M. and P. Brown. Object-Relational DBMS: Tracking the Next Great Wave. San Francisco: Morgan Kaufmann, 1999.

21. Trujillo, Juan, Manuel Palomar, Jaime Gomez, and Il-Yeol Song. "Designing Data Warehouses with OO Conceptual Models." IEEE Computer 34, no. 12 (December, 2001): 66-75.

22. Ullman, Jeffrey D., Jennifer Widom, and Jennifer D. Widom. A First Course in Database Systems. New Jersey: Prentice Hall PTR, 2001.

23. Wixom, Barbara, Paul Gray, and Hugh J. Watson. "Introduction to the Minitrack on Data Warehousing," Proceedings of the 34th Hawaii International Conference on System Sciences – 2001 (HICCSS'01), Institute of Electrical and Electronics Engineers, Hawaii, 2001.