Open Research Online

The Open University's repository of research publications
and other research outputs

The representation of time in data warehouses

Thesis

How to cite:

Todman, Christopher Derek (1999). The representation of time in data warehouses. PhD thesis The Open University.

For guidance on citations see FAQs.

© 1999 The Author

Version: Version of Record

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data policy on reuse of materials please consult the policies page.

oro.open.ac.uk

UNRESTRICTED

The representation of time

in

data warehouses

Christopher Derek Todman BSc. (Hons)

A thesis submitted for the degree of
Doctor of Philosophy

The Open University
Department of Computing
(faculty of Mathematics and Computing)

June 1999

Date 1st June 1999

AUTHOR'S NO : M2L104S4
DATE OF SUBMISSION: 28 MAY 1999
DATE OF AWARD: 30 SEPTEMBER 1999

The representation of time in data warehouses

C.D Todman

Abstract

This thesis researches the problems concerning the specification and implementation of the temporal requirements in data warehouses. The thesis focuses on two areas, firstly, the methods for identifying and capturing the business information needs and associated temporal requirements at the conceptual level and; secondly, methods for classifying and implementing the requirements at the logical level using the relational model.

At the conceptual level, eight candidate methodologies were investigated to examine their suitability for the creation of data models that are appropriate for a data warehouse. The methods were evaluated to assess their representation of time, their ability to reflect the dimensional nature of data warehouse models and their simplicity of use. The research found that none of the methods under review fully satisfied the criteria.

At the logical level, the research concluded that the methods widely used in current practice result in data structures that are either incapable of answering some very basic questions involving history or that return inaccurate results.

Specific proposals are made in three areas. Firstly, a new conceptual model is described that is designed to capture the information requirements for dimensional models and has full support for time. Secondly, a new approach at the logical level is proposed. It provides the data structures that enable the requirements captured in the conceptual model to be implemented, thus enabling the historical questions to be answered simply and accurately. Thirdly, a set of rules is developed to help minimise the inaccuracy caused by time.

A guide has been produced that provides practitioners with the tools and instructions on how to implement data warehouses using the methods developed in the thesis.

Acknowledgements

To Mike Newton and Pat Hall, my supervisors.

And especially to my wife, the other Chris Todman,
for your patience and support.

The responsibility for this work is my own.

Contents

List of figures

Number *Page*

1 Introduction

1.1 Background

The objective of this chapter is to provide a general introduction to the context of the research problem that is the principal subject of this thesis. It also describes the objectives of the thesis and outlines the strategy for the research together with the structure and contents of the thesis.

The principal subject of the thesis is data warehousing. A data warehouse is a special kind of database which, in recent years, has attracted a great deal of interest in the information technology industry.

Data warehousing has become so popular in industry that it is cited, by Violino (1998), as being the highest priority post-millennium project of more than half of IT executives. It is estimated, by the Palo Alto Management Group (see Menefee (1998)), that in 1997 $15 billion was spent on data warehousing world wide. Recent forecasts (Business Wire August 31 1998) expect the market to grow to around $113 billion by the year 2002. A study carried out by the META Group in 1996 (see Meyer & Cannon (1998)) has found that ninety five percent of the companies surveyed intended to build a data warehouse.

Data warehousing is being taken so seriously by the industry that the Transaction Processing Council (TPC), which has defined a set of benchmarks for general databases, has introduced a benchmark specifically aimed at data warehousing applications as described by Packer (1998). As a further indication of the 'coming of age of data warehousing', a consortium has developed an adjunct to the TPC benchmark called 'The data warehouse challenge' (see Winter (1998)) as a means of assisting prospective users in the selection of products.

The benefits of building a data warehouse can be significant. Mott (1998) says increasing knowledge within an organisation about customers' trends and business can provide a significant return on the investment in the warehouse. Kight (1996) cites savings made by Union Pacific of $1 million per annum in sales tax, $2 million by being able to move competitors rolling stock 'off-line' and a further $1.5 million from information enabling them

to reduce their accounts receivable cycle. According to Foley (1996), another organisation, MCI, found it could cut the cost of customer lead generation by ninety percent.

The proliferation of these systems has resulted in a high degree of frenetic development activity. Database designers and system integrators have attempted to take on the implementation of data warehouse applications using the same skills, methodologies and tools as they were using for operational systems. However, there is an important difference between operational applications and data warehouses that has not been fully appreciated by data warehouse practitioners. It is that data warehouses are temporal applications and, therefore, require true temporal support from the database management systems in which they are implemented. The absence of such support has gone largely unnoticed and scant regard is given to this serious issue. The result is that many data warehouses return inaccurate results from queries involving time. In data warehouses, virtually every query involves time and this means that the result of every query is suspect. Organisations are increasingly relying on the information from data warehouses to provide support for decision making. The concern is that the decisions are being made with the assistance of faulty information.

The main hypothesis of this thesis, therefore, is that the representation of time in data warehouses is generally not adequate and this can lead to inaccuracy in the results obtained from queries.

The degree to which this is a problem varies from one organisation to another and in many cases the error may be small enough to have no significant effect on the value of the results. Unfortunately, most organisations do not fully recognise the issue and almost certainly cannot assess the level of accuracy of the data held in the database. The need for accuracy is recognised by Surveyer (1998) who says that the

'Accuracy of data and information is as critical as their presentation'.

Foley (1996) goes further saying that accuracy is 'a must' and that if the users do not trust the accuracy of the data then the data warehouse is 'garbage'.

In this thesis I will show that, even though there is no comprehensive support for time in database management systems today, it is possible to increase significantly the level of accuracy of query results from a data warehouse.

As data warehousing is a relatively new type of database application, it has received very little attention from the academic community. It has been difficult to find serious research in data warehousing generally and I have found none in the field of time in data warehousing. Two papers recognise that time is an issue, Widom (1995) and Sakaguchi (1996), both insisting that it does justify attention from researchers. However, most of the existing knowledge has been created by practitioners in the field and much of the literature is in the form of 'white' papers that are often produced by consultants, and some academics, who have been commissioned by the owners of software packaged systems. The academic rigour, in these cases, often gives way to commercial pressure in the sense that: *'who pays the piper calls the tune'*. In addition to the white papers there exists a large amount of published material, mostly in the form of trade journal articles and books. Generally, the published material deals with the high level issues and not with the detailed technical problems. There are a small number of books that are regarded as seminal in nature. Within these, relatively little space is allocated to the problems associated with time. Snodgrass (1998) agrees that trade publications and books have provided no guidance or techniques for managing time in data warehouses.

However, there has been an enormous amount of research conducted into the field of temporal databases. This research has resulted in the creation of a set of standard terms and a general understanding that has been published in the Consensus Glossary of Temporal Database Concepts by Jensen et al (1994).

This extensive research does provide some help in the data warehousing field in that there are:

- A standard set of terms
- A generally accepted set of temporal query functions
- Some ideas regarding the physical implementation

At the present time (Spring 1999), development by the database management system manufacturers has not progressed to the point where temporal database management systems are generally available and the assumption is made that the widespread introduction of such systems is still some way off. Practitioners have to find ways of implementing data warehouses with adequate support for time using the tools at their disposal today. Thus I have concluded that to pursue a solution that depended on the research so far undertaken into temporal databases would not result in a method that could be adopted by practitioners today. My research, therefore, will encompass the temporal database research only insofar as it provides assistance in the understanding of temporal issues.

Most data warehouses are implemented using relational database management systems and my investigation is largely focused on this technology. However, there is a relatively new type of database management system used in data warehousing and some consideration will be given to this. It exists under the general heading of *'On-line Analytical Processing'* (OLAP). The general term OLAP has become synonymous with data warehouses. There are many OLAP products on the market, in many shapes and forms, which are all derivatives of the same basic design. The most common derivatives are:

MOLAP – Multi-Dimensional OLAP: This is usually a product supported by a proprietary multi-dimensional database management system (MDDBMS) that is designed to support a dimensional schema.

ROLAP – Relational OLAP: This is a dimensional model implemented in a relational database (RDBMS). This is the most common form of implementation.

HOLAP - Hybrid OLAP: This means Multi-Dimensional *and* Relational OLAP: This kind of product uses both types of DBMS. The high level, summarised data is stored using the MDDBMS but the system allows a 'drill down'[1] facility into the RDBMS to obtain more detailed results.

[1] Drilling down is the capability to further analyse a set of results to a finer level of grain. The concepts of drill down, drill up and drill across are described by R Kimball (1996i) and further in Kimball (1996b)

There are views as to which is preferable and these are described by Youngworth (1995), Calloway (1995) and Elkins (1998a and 1998b). This thesis is not concerned with approaches at this level. It is sufficient to recognise that they all implement a dimensional model.

The databases used by these products are often referred-to as Data Marts. A data mart is usually regarded as a small, perhaps departmental, data warehouse. It may be a stand-alone system, or it may contain a subset of a much larger data warehouse.

For the purpose of this thesis, the terms data warehouse and data mart can be used synonymously.

Data warehousing is the first major popular temporal application that we have encountered and this has introduced a new set of problems that practitioners are not equipped to deal with. The academic research that has been carried out so far has not resulted in the provision of any guidance that is appropriate for adoption by practitioners. There is benefit to be gained, therefore, in according the temporal issues in data warehousing an academic perspective that has so far been lacking. In view of this, the purpose of my research is to:

1. Examine how data warehouses are affected by time and understand the specific requirements for the treatment of time in data warehouses.

2. Examine the methods currently used to satisfy the requirements and identify the useful components as well as the flaws in the methods.

3. Having learnt from existing methods, explore ways of overcoming their flaws by enhancing them and creating new methods where needed.

There are one or two practitioners in the field who have invented methods for providing limited solutions to some of the temporal problems using the current technology. In the absence of alternatives, these methods have been broadly adopted by the industry. The prescribed solutions provide partial representation for time but are presented in such a way as to, not deliberately, mislead practitioners into thinking that all the temporal aspects of data warehouses have been taken into account. Close examination of these techniques shows that they provide some advantages but also some quite serious disadvantages. They tend to be

applied in a somewhat ad hoc fashion that indicates that there exists a general lack of comprehension of the real underlying issues. These issues are explored in the next chapter of this thesis.

There are, essentially, two main areas where the subject needs to be addressed. Broadly, these are at the conceptual and logical levels of developing a data warehouse.

At the conceptual level, there is no data modelling method that is specific to data warehousing. If there were, it would greatly increase the chances of capturing the temporal requirements of the application. In this thesis I review some of the modelling methods that have emerged in an attempt to provide support for time and which have been developed as part of the research into temporal database management systems. The purpose of this review is to assess the viability of adopting such methods for the modelling of data warehouses.

At the logical model level, data warehousing has received more attention and the methods in use are subjected to a review and critique.

As a practitioner in this field, I am concerned about the quality of what is being delivered today. We must seek to improve the levels of accuracy of data warehouses. My research has shown that, due to the time lag between operational systems and data warehouse processes, some degree of inaccuracy is inevitable. However, where we can improve accuracy, we should do so.

The methodology developed in this thesis will be made available in the form of a practitioner's guide that is designed to raise the general level of understanding of the problems and to prescribe an approach to designing data warehouses that encompasses a complete set of solutions.

1.2 Terminology

Some of the terms used in the thesis will now be defined.

1.2.1 Data models and schema

In this thesis, when describing data models, I have adopted the following definitions:

- A ***conceptual*** data model represents reality at a high level of abstraction. It has the property of being easy to understand and interpret. The purpose of a conceptual model is to describe the information that is required to be stored in the database from the point of view of the users or owners of the data. It is independent of any database management system or implementation requirements. For instance, the entity relationship notation defines the means of representing a conceptual data model.

- A ***logical*** data model is a definition of data containing the data descriptions that can be processed by a computer. These models are easily mapped to the physical structure of the database. An example of a logical data model can be produced using, say, relational theory.

- A ***physical*** data model describes the storage structures and access methods used to gain access to the data. The physical model is, therefore, tied to a particular database management system.

1.2.2 Valid time and transaction time

The definitions for valid time and transaction time are taken from The Consensus Glossary of Temporal Database Concepts that was compiled by C.S Jensen et al (1994). The ***valid time*** associated with the value of, say, an attribute is the time when the value is true in modelled reality. Such values may be associated with:

1. A single instant, which is defined to be a time point on an underlying time axis. An ***event*** is defined as an instantaneous fact that occurs at an instant.

2. Intervals (periods) of time. An interval is defined to be the time between two instants.

The valid time is normally supplied by the user.

The ***transaction time*** associated with the value of, say, an attribute records the time at which the value was stored in the database and is able to be retrieved. Transaction times are system

generated and may be implemented using transaction commit times. Transaction times may also be represented by single instants or time intervals.

1.3 A description of data warehousing

The concepts of data warehousing and dimensional analysis are now explored with the aim of demonstrating the special characteristics of data warehouses, their development and use.

1.3.1 Introduction to data warehousing

A data warehouse is a particular kind of database that relates to a branch of a general business subject known as decision support as defined below by Sprague and Carlson (1982). In order to understand data warehousing, therefore, it is necessary to understand the purpose of decision support systems in general.

The purpose of a decision support system (DSS) is to provide decision makers in organisations with information. The information advances the decision maker's knowledge in some way so as to assist them in making decisions about the organisation's policies and strategy.

A DSS tends to have the following characteristics:

- They tend to be aimed at the less well structured, under-specified problems that more senior managers typically face.

- They attempt to combine the use of models or analytic techniques with traditional data access and retrieval functions.

- They possess capabilities that make them easy to use by non-computer people interactively.

- They are flexible and adaptable enough to accommodate changes in the environment and decision making approach of the user.

The job of a DSS is usually to provide a factual answer to a question phrased by the user.

For instance: a sales manager would probably be concerned if her actual product sales were falling short of the target set by her boss. The question she would like to be able to ask might be:

Why are my sales not meeting my targets ?

This is a very complex question that cannot be directly translated into a database query.

Her questioning has to be more systematic such that the DSS can give factual responses. So the first question might be:

For each product, what are the cumulative sales and targets for the year ?

A DSS would respond with a list of products and the sales figures. It is likely that some of the products are ahead of target and some are behind. A well constructed report might highlight the offending products to make them easier to see. For instance, they could be displayed in red, or flashing. She could have asked:

What are the cumulative sales and targets for the year for those products where the actual sales are less than the target ?

Having discovered those products that are not achieving the target, she might ask what the company's market share is for those products, and whether the market share is decreasing. If it is, maybe it's due to a recently imposed price rise.

The purpose of the DSS is to respond to ad hoc questions like these, so that the user can ultimately come to a conclusion and make a decision.

A major constraint in the development of DSS is the availability of data. That is, having access to the appropriate data when it is needed. The introduction of sophisticated DBMSs has certainly eased the problems caused by traditional applications but, nonetheless, unavailability of data still persists as a problem for most organisations. The main reasons for this are as follows. Most organisations evolve over time. As they do, the application systems increasingly fail to meet the functional requirements of the organisation. The result is that the applications are continually being modified in order to keep up with the ever changing business. There

comes a time in the life of almost every application where it has been modified to the point where it becomes impossible or impractical to modify it further. At this point a decision is usually made to redevelop the application. When this happens, it is usual for the developers to take advantage of whatever improvements in technology have occurred during the life of the application. For instance, the original application may have used indexed sequential files because this was the most appropriate technology of the day. Nowadays, of course, most applications obtain their data through an RDBMS.

However, most large organisations have dozens or even hundreds of applications. These applications reach the end of their useful lives at various times and are redeveloped on a piecemeal basis. This means that, at any point in time, an organisation is running applications that use many different types of software technology.

Further, large organisations usually have their systems on diverse hardware platforms. It is very common to see applications in a single company spread over:

- A large mainframe
- Several mid-range multi-processor machines
- Networked and stand-alone PCs

This type of scenario, which is a typical real world situation, causes problems when we try to provide information about the organisation because a DSS may require to access information from many of the applications in order to answer the questions being put to it by its users. The problem is one of integration. Gaudin (1996) recognises this as an issue that affects most organisations when building data warehouses. In order for useful information to be presented to the decision makers, it must first be extracted from the source systems where it is held. These systems, as has been described, can be quite disparate in nature. Their processing cycles, granularity of time and level of detail, as well as differences in semantics and format makes it very difficult to meld the data in a way that makes it easier for decision makers to access.

1.3.2 The Wine Club example

The described features can be illustrated by examining the operation of a fictitious organisation. A case study database has been created that will be used to provide examples throughout this thesis.

The example used in the case study has been chosen on the basis that no real business or technical knowledge should be assumed, or needed, on the part of the reader. However, it is important that the case study application is able to provide realistic examples of business issues that must be solved.

The organisation is a wine club. It is called, with great originality, the Wine Club. It has customers (Members) and products (Wines). In addition the wines are obtained from suppliers and delivered via shippers. There is a supporting database of information in the form of a data warehouse. Some of the data from the data warehouse is printed in the appendix to this document.

The club's data can be modelled, by an Entity Relationship (ER) diagram (Chen (1976) and Batini et al (1992)).

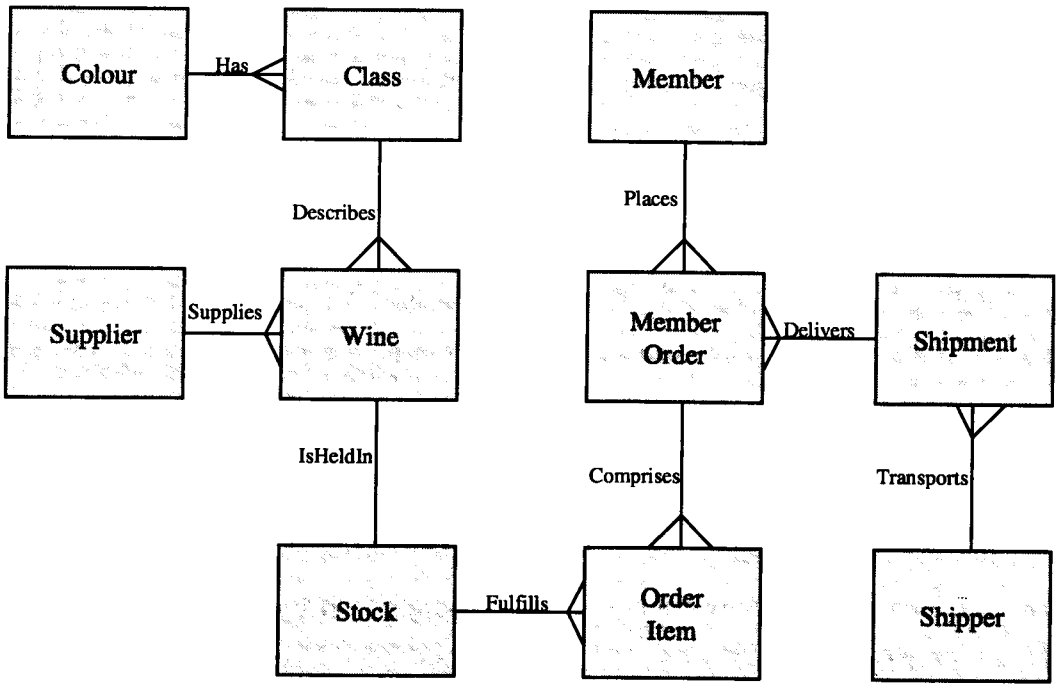


Figure 1.1 Entity Relationship diagram for the Wine Club

Entity Descriptions

Class (ClassCode, ClassName, Region)

Colour (ColourCode, ColourDesc)

Member (MemberCode, MemberName, MemberAddress)

MemberOrder (OrderCode, OrderDate, ShipDate, Status, TotalCost)

OrderItem (OrderCode, ItemCode, Quantity, ItemCost)

Shipment (ShipCode, ShipDate)

Shipper (ShipperCode, ShipperName, ShipperAddress, ShipperPhone)

Stock (LocationCode, StockOnHand)

Supplier (SupplierCode, SupplierName, SupplierAddress, SupplierPhone)

Wine (WineCode, Name, Vintage, ABV, PricePerBottle, PricePerCase, CostPerBottle)

The Wine Club has the following application systems in place:

Member Administration. This enables the club to add new members. This is particularly important after an advertising campaign, typically in the Sunday Colour Supplements, when many new members join the club at the same time. It is important that the new members' details, and their orders, are dealt-with promptly in order to create a good first impression. This application also enables changes to a member's address to be recorded, as well as removing ex-members from the database. There are about 100,000 active members.

Stock Control. The goods inwards system enables newly arrived stock to be added to the stock records. The club carries about 2,200 different wines from about 150 suppliers.

Order Processing. The directors of the club place a high degree of importance on the fulfilment of members' orders. Much emphasis is given to speed and accuracy. It is a stated policy that orders must be shipped within ten days of receipt of the order. The application systems that support order processing are designed to enable orders to be recorded swiftly so that they can be fulfilled within the required time. The club processes about 750,000 orders per year, with an average of 4.5 items per order.

Shipments. Once an order has been picked, it is packed and placed in a pre-designated part of the dispatch area. Several shipments are made every day.

The Club's systems have evolved over time and have been developed using different technologies. The order processing and shipments systems are based on indexed-sequential files accessed by COBOL programs. The membership administration is held on a relational database. All these systems are executed on the same mid-range computer. The stock control system is a software package that runs on a PC network.

There is a general feeling amongst the directors and senior managers that the club is losing its market share. Within the last three months, two more clubs have been formed and their presence in the market is already being felt. Also, recently, more members than usual appear to be leaving the club and new members are being attracted in fewer numbers than before.

The directors have held meetings to discuss the situation. The information upon which the discussions are based is largely anecdotal. They are all certain that a problem exists but find it impossible to quantify. They also know that the information that would help them passes through their systems and should be available to answer questions. In reality, however, whilst it is not too difficult to get answers to the day-to-day operational questions, it is almost impossible to get answers to more strategic questions.

The distinction between the terms '**strategic**' and '**operational**', insofar as information is concerned, is a general issue in decision support. Strategic matters deal with *planning and policy making*. For instance, in the Wine Club example, the decision as to when a new product should be launched would be regarded as a strategic decision. Strategic decisions are often linked to strategic business goals. Wallace's (1994) and Inmon and Hackathorn's (1994) work provide the practitioner's view as to the typical business goals of users of decision support systems:

- Increasing market share
- Reducing costs and expenses
- Increasing revenue
- General competitive business advantage

These are strategic goals. According to Foley (1996), examples of strategic decisions pertaining to other types of organisation include:

- The affect of reducing the direct sales force in favour of more telemarketing.
- The introduction of very cheap off-peak tariffs to attract telephone callers away from the peak times, rather than install extra equipment to cope with increasing demand.
- The opening of supermarket stores on Sundays.
- A general twenty percent price reduction for one month in order to increase market share.

Whereas strategic matters relate to planning and policy, operational matters are generally more concerned with the *day-to-day running* of a business or organisation. Operations can be regarded as the *implementation* of the organisation's strategy (its policies and plans).

According to Foley, the day-to-day ordering of supplies, satisfying customers' orders and hiring new employees are examples of operational procedures. These procedures are usually supported by computer applications and, therefore, they must be able to provide answers to operational questions such as:

- Has a customer been invoiced ?
- On which items are we out of stock ?
- What is the status of a particular order ?

Typically, operational systems are quite good at answering questions like these because they are questions about the situation as it exists currently.

Returning to the Wine Club, the kind of questions the directors may wish to ask are:

1. Which product lines are increasing in popularity and which are decreasing ?
2. Which product lines are seasonal ?
3. Who are the members who place the same orders on a regular basis ?
4. Are some products more popular in different parts of the country ?
5. Do members tend to purchase particular classes of product ?

The five questions above are clearly not 'current-state' types of questions and, typically, operational systems are *not* good at answering such questions.

The reason for this lies in the nature of operational systems. They are developed to support the operational requirements of the organisation. An examination of the operational systems

of The Wine Club reveals what they actually do. Each application's role in the organisation can usually be expressed in one or two sentences.

The membership administration system contains details of current members. The stock control system contains details of the stock currently held. The order processing system holds details of members' orders that have yet to be fulfilled and the shipments system records details of fulfilled orders awaiting delivery to the members.

The use of words like 'details' and 'current' in those descriptions underline the 'current state' nature of operational systems. It could be said that the operational systems represent a 'snapshot' of an organisation at a point in time. The values held are constantly changing. At any point in time, many inserts, updates and deletes may be executing on all, or any, parts of the systems. If any system were frozen momentarily, then they would provide an accurate reflection of the state of the organisation at precisely that moment. One second earlier, or one second later, the situation would be different.

A re-examination of the five questions that the directors of The Wine Club need to ask in order to reach decisions about their future strategy shows that they have something in common. Each of the five questions is concerned with *sales of products over time.* One way of assessing whether a product line is increasing or decreasing in popularity is to trace its demand over time.

If the order processing data was held in a relational database, we could devise an SQL query such as:

```
Select Name, Sum(Quantity), Sum(Quantity * ItemCost) Sales
    from MemberOrder a, OrderItem b, Wine c
    where a.OrderCode = b.OrderCode
    and a.WineCode = c.WineCode
    and OrderDate = <today's date>
    group by Name
```

Figure 1.2 Query to calculate sales quantities and values

If this query were to be executed at the end of the day, it would return the value of all the orders received for the day.

Generally speaking, this is useful information but it is not enough by itself. The value of a piece of information, in this context, is only realised when it is compared to something else. What is required is to examine the trend over (say) the last six months, or to compare this month with the same month last year.

A solution could be that the query is executed every day and the results for each day are appended into a table. That way, over time, it is possible to build up the historical data that is needed. This could be the beginning of a data warehouse. What I have tried to show is that one of the main properties of data warehouses, that distinguishes them from operational systems, is that they always store historical data. Data Warehouses possess other properties as well. W.H Inmon has been credited with the title of 'The father of Data Warehousing' by Kimball (1996c) and Meyer & Cannon (1998). Inmon's (1992) definition of a Data Warehouse has become the de facto industry accepted definition.

He says that a data warehouse is:

> ***'A subject oriented, non volatile, integrated and time variant collection of data in support of management's decisions' (page 29)***

There are other definitions, Devlin (1997) defines it as:

> ***'A single, complete and consistent store of data obtained from a variety of sources and made available to end users in a way they can understand and use in a business context'. (page 20)***

He does agree with Inmon that decision support data is organised around subjects like 'Sales Summary' and 'Market Analysis'. This is consistent with Inmon's property of 'subject orientation'. He also agrees that integration is important because he says we have to ensure that the data is 'consistent across the enterprise'. Further, he accepts that time variance is important because:

> ***'The need to access historical data is one of the primary incentives for adopting the warehouse approach'.(page 104)***

Sen and Jacob (1998) conducted a survey of researchers, academics and vendors and found broad agreement that data warehouses are built in the interests of business decision support and will contain historical data from a number of operational databases. They go on to say that data warehouses are multi-dimensional in nature.

The other views are quite similar to Inmon's and, as Inmon's definition is a little more precise, I will adopt it for the purposes of this thesis. I shall now describe the terms more fully.

Subject oriented means that the data is organised around a single focal subject area such as Sales. This distinguishes data warehouses from operational applications, such as order processing, where the data is organised around business applications and which makes them *application oriented.* The subject area in a data warehouse can usually be defined as the state of a particular entity at some point in its life cycle. The achievement of this state is triggered by an event. For instance, the production of an invoice may cause a transition in the state of an order to 'invoiced'. In a telecommunications system, the re-placement of the handset, when detected by the exchange, may change the state of the call to 'completed'. In a university, the attendance of a student at an examination may result in the state being set to 'attended'. Once the event has occurred, and the state achieved, the entity can be captured and placed into the warehouse. It is usually important to focus on the event rather than the state because states can be triggered by more than one event. For instance, the requirement might be to record the value of an order into the data warehouse when the order is invoiced. That is, when the state changes to 'invoiced'. Under normal circumstances, an order changes its state to 'invoiced' once the signed delivery note has been entered and its value would be captured at that point. If the invoice was disputed by the customer, then the event of recording the dispute might change the state of the order to 'in dispute'. Once the dispute is resolved, the resolution event

might cause the order to return to the 'invoiced' state. Clearly, the required processing, insofar as the data warehouse is concerned, is different when the invoice changes its state from, say, 'delivered' to 'invoiced' than when it changes from 'disputed' to 'invoiced'.

The five queries that the directors have identified as examples of the types of questions they would like to ask of their data are concerned with sales of products to members over time. The subject area focus is clearly 'Sales'.

Sales is the most common subject area for data warehousing. The definition of a sale is a good example of what has just been described because different roles in an organisation will take a different view as to which event actually triggers the sale. An accountant may say that a sale has occurred when an invoice has been produced. A salesman might say that a sale has occurred when an order is received. A warehouse manager might say that a sale has occurred when a delivery note is signed by the customer.

The point is that the term 'Sale' is ambiguous and needs to be defined within the organisation to avoid confusion. It is highly unlikely that there is an entity called 'Sale' anywhere in the corporate data model despite the popularity of sales in data warehousing. In practice, the users of the data warehouse have to be led through an exercise in understanding the semantics of their data so that the entity can be focused upon, and the data extracted, when the appropriate point in the entity life cycle has been reached. This is often quite difficult to achieve, especially if the application that processes the data has no interest in the fact that this important state has been achieved. In these cases some modification to the application, or the database, has to be made.

Non volatile means that the data, once placed in the warehouse, is not usually subject to change. While it is possible, and necessary, to add new data to the warehouse on a regular basis it is not expected that the data will change after it has been loaded. Anyone who is using the database has confidence that a query will always produce the same result no matter how often it is run. In contrast, operational databases are extremely volatile in that they are constantly changing. A query is unlikely to produce the same result twice if it is accessing tables that are frequently updated. Kimball (1996a) refers to the 'twinkling' nature of

operational databases. The expectation of users of these systems is also different in that they expect to see the very latest information.

Integrated means the data is combined from different sources so that it is consistent. There are two types of integration in a data warehouse. The first type is called *format integration* which ensures that all data items of the same type share the same physical format. For instance, it may be important to ensure that dates are always stored in the same format such as 'yyyy/mm/dd'. Integration is a problem for most organisations particularly where there are many different types of technology in use. Some differences are quite fundamental such as the character set. Most systems use the ASCII (American Standard Code for Information Interchange) character set but some do not. IBM, for instance, bases all of its mainframe systems and many of its mid-range systems on a totally different character set called EBCDIC (Extended Binary Coded Decimal Interchange Code). So the letter 'P' has a value of 80 in ASCII but is 215 in EBCDIC (the character with a value of 80 in EBCDIC is '&'). The word 'Pool' in ASCII translates to '&??%' in EBCDIC and it's difficult to imagine anything less integrated than this. Other differences are more subtle: most DBMSs have a 'Date' data type (although the storage format is different from one DBMS to another) whereas access methods such as indexed sequential have no such facility. Even more subtle differences occur within different applications within the same technology. This occurs where, for instance, one application designer decides to hold addresses as five columns of twenty-five characters each, whereas another might use a *Varchar(100)* format.

The second type of integration is called *semantic integration.* This ensures that consistency in the meaning of data is maintained. The consolidation of data from disparate source systems inevitably leads to semantic differences that have to be resolved before the data is stored in the data warehouse.

Time variant means that historical data is recorded. Almost all queries executed against a data warehouse have some element of time associated within them. I have already established that most operational systems do not retain historical data. It is almost impossible to predict what will happen in the future without observing what happened in the past. A data warehouse helps to address this fundamental issue by maintaining a historical dimension to the data taken from the operational databases. Inmon (1992) describes data stored in a data

warehouse as being accurate as of some moment in time. This is in contrast to operational data that is said to be accurate as at the moment of access.

1.3.3 Dimensional Models

The main approach to data warehouse design is to develop and implement a 'Dimensional Model'. This has given rise to Dimensional Analysis (sometimes generalised as Multi-Dimensional Analysis).

It was noticed quite early-on when data warehouses started to be developed, that whenever decision makers were asked to describe the kinds of questions they would like to get answers to regarding their organisations, they almost always wanted the following:

- Information presented, initially, in a summarised form. The information should be grouped into, say, 'total sales' rather than displayed as individual items. There is a further requirement to be able to break the summaries into more detail or to 'drill down', as Kimball (1996b) would say.

- Analysis of the summarised information across their own organisational components such as 'departments' or 'regions'.

- To enable the information to be displayed in both graphical and tabular form.

- The capability to view their information 'over time'.

The Wine Club provides an example. The directors might wish to see a report showing Sales by Product, or a report showing Sales by Member, or even Sales by Product by Member

Figure 1.3 below shows an example Wine Club report of sales by product.

Product Name	Quantity Sold (Cases)	Cost Price	Selling Price	Total Revenue	Total Cost	Gross Profit
Chianti	321	26.63	42.95	13,787	8,548	5,239
Bardolino	1,775	15.10	31.35	55,646	26,802	28,844
Barolo	275	46.72	70.95	19,511	12,848	6,663
Lambrusco	1,105	23.25	41.45	45,802	25,691	20,111
Valpolicella	2,475	12.88	32.45	80,313	31,878	48,435

Figure 1.3: The Wine Club - Analysis of sales by product for July

Figure 1.3 shows a report of sales for July, analysed by product. There is a general point here in that the inclusion of a temporal dimensional constraint is always required as it gives meaning to the results, allowing them to be compared to some other period of time, or to another set of products for the same period of time.

The 'dimensional' approach led Ted Codd to make the following observation:

> ***'There are typically a number of different dimensions from which a given pool of data can be analysed. This plural perspective, or Multi-Dimensional Conceptual View, appears to be the way most business persons naturally view their enterprise'.*** Codd et al (1993)

The approach is to determine, after consultation with the appropriate decision makers in an organisation, which is the *subject area* that they are most interested in, and what are the most important *dimensions of analysis.*

The subject area reflects the subject oriented nature of the warehouse. In the Wine Club example above, the subject area would be Sales. The dimensions of analysis would be Members and Products. The requirement is to analyse sales by product and by Member and is depicted in the three dimensional cube in Figure 1.4 below:

The diagram shows Sales (the shaded area) having axes of Member, Product and Time

Traditionally, in data warehousing, a time dimension is always included. The purpose of the time dimension is to enable the facts to be grouped by time. The time dimension is joined to the facts by the foreign key implemented using the time code attribute. The user places time

groupings into queries by adding attributes such as 'season' and 'year' into the 'group by' clause. The query processor is then able to group the result set into the required aggregations.

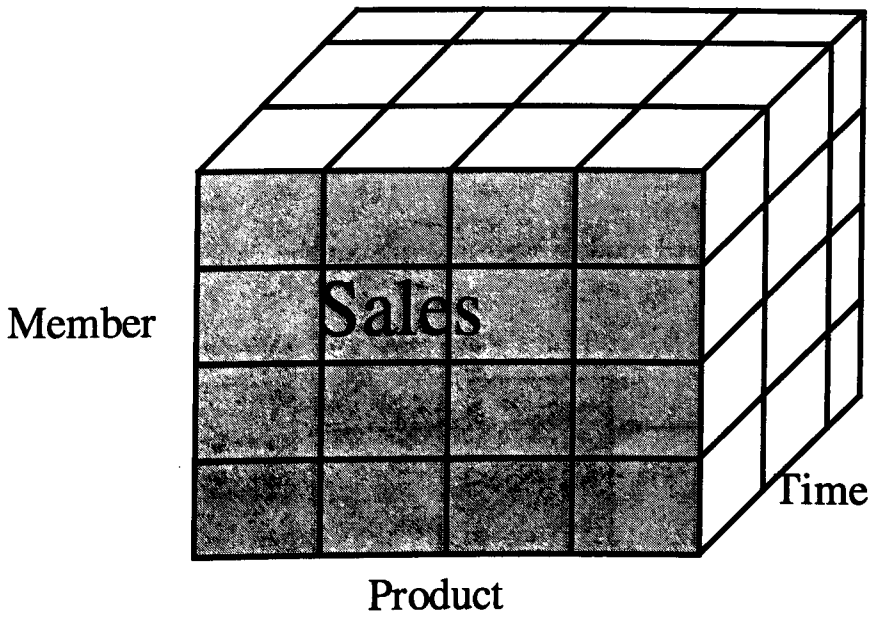


Figure 1.4 Multi Dimensional Cube

This means that Sales can be analysed by Member by Product over Time. So each element of the cube (each mini-cube) contains a value for sales to a particular member, of a particular product, at a particular point in time. Although the example in Figure 1.3 used 'July' which, intuitively, might be regarded as a interval or period of time, it is really a question of granularity. For the purposes of the report, the time granularity is 'month'.

The multi dimensional cube above shows sales as the subject with three dimensions of analysis. There is no real limit to the number of dimensions that can be used in a dimensional model although there is, of course, a limit to the number of dimensions we can draw !

The directors of The Wine Club need answers to questions about sales. Looking back at the five example questions, they all concerned sales. Sales by product. Sales by member. Sales by area. As in the example above, the subject area for their data warehouse is clearly 'Sales'.

So, in the example, the dimensions of analysis are:

- Product
- Member
- Sales area of member
- Time

As we cannot draw four-dimensional models, we can represent the dimensional conceptual schema in the following way:

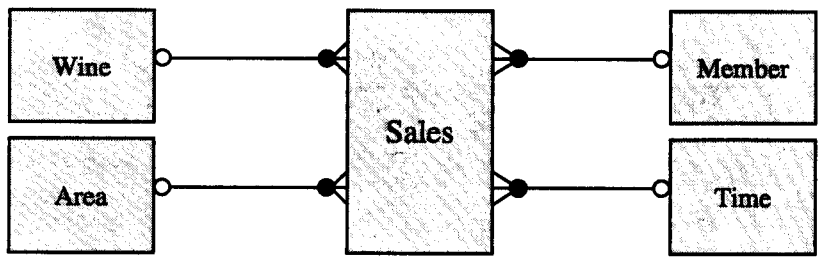


Figure 1.5 Dimensional Star Schema

The above diagram is often referred-to as a *Star Schema* because the diagram loosely resembles a star shape. The subject area is the centre of the star and the dimensions of analysis form the points of the star. The subject area is often drawn long and thin because the table itself is usually long and thin in that it contains a small number of columns but a very large number of rows.

The generic star schema is the most commonly used diagram for depicting dimensional data warehouses. It does have a drawback in that, in reality, dimensions of analysis that can be represented by a single entity are rare. Usually, the dimensions are organised into hierarchical structures.

In the Wine Club, for instance, the 'Wine' dimension is organised by wine growing region and also by supplier. This will enable the directors to group their query results into regions and to assess which of their suppliers provides the most popular or the most keenly priced products. Similarly, members are organised into different categories to assist with the query formulation.

This means that the dimensions are normalised into hierarchical structures. When this happens, the true star shape begins to disappear and the model takes on more of the appearance of a 'snowflake'. Hence the term 'Snowflake Schema' is generally used to describe such diagrams.

The snowflake model for the Wine Club is shown in Figure 1.6:

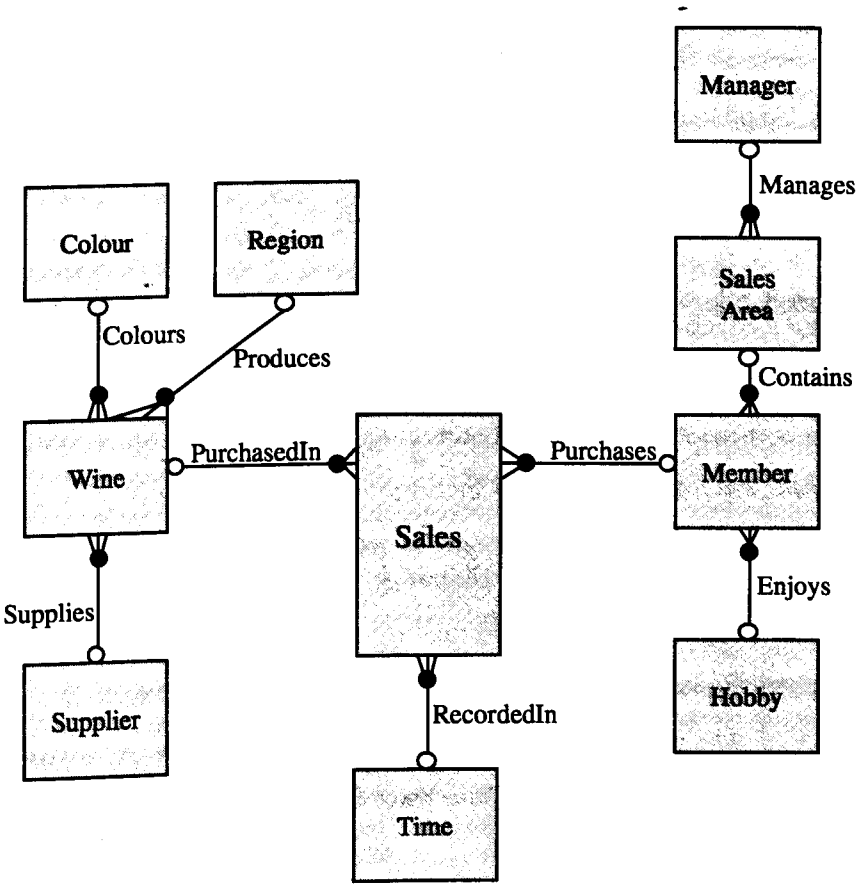


Figure 1.6 Dimensional (Snowflake) schema for the Wine Club

The descriptions of the fact table and each of the dimensions are as follows:

Entity Descriptions

Sales_Area(Sales_Area_Code, Manager_Code, Sales_Area_Name)

Manager(Manager_Code, Manager_Name)

Colour(Colour_Code,Colour)

Time(Time_code, Day_Name, Week_End, Week, Month, Month_Name, Season, Year)

Hobby(Hobby_Code, Hobby_Name)

Member(Member_Code, Member_Name, Member_Address,Sales_Area_Code,
Hobby_Code,Date_Joined)

Region(Region_Code, Region_Name, Country)

Sales(Member_Code, Wine_Code, Time_Code, Quantity,Value)

Supplier(Supplier_Code, Supplier_Name, Supplier_Address, Phone)

Wine(Wine_Code, Supplier_Code, Wine_Name, Colour, Vintage, ABV, Region,

Bottle_Price, Case_Price, Bottle_Cost)

In some approaches using ER diagrams, entities do not include foreign keys in a conceptual model such as this. However, in a dimensional model where the impact of changes is to be assessed, changes in the values of attributes that implement relationships are as important as changes to the value of other attributes. It is felt, therefore, that this requirement overrides the other approaches and the foreign key attributes have been included.

The star and snowflake schema approach are described in detail in much of the literature including Kimball (1996j), Inmon et al (1997), Devlin (1997), Meyer and Cannon (1998), Poe et al (1997), Hammergen (1998), Dodge and Gorman (1998), Snodgrass (1997), Menninger (1995b), Fadlalla (1996), Demarest (1998) and McClanahan (1997). Most authors advocate the use of this kind of design for part, if not all, the data warehouse. It is generally accepted that the best approach is to produce simple models that users can relate to their organisation, as described by Atre (1998a). Others such as Rudin (1996) and Rigney (1996) recommend caution in that alternative approaches should be considered. Even advocates of dimensional schemas such Devlin say that we should not 'force' a star when it may not be the best solution. There is just one article, by Gardner (1998) that states that any architecture not based on the

third normal form model can cause the failure of a data warehouse project because the needs of the warehouse are constantly changing. There exists a general belief amongst practitioners that dimensional models are, by definition, not normalised. This is not necessarily the case although, for usability and performance reasons, a degree of 'strategic' denormalisation is often recommended by some practitioners, especially Kimball.

On balance, the preference weighs heavily in favour of the star schema and, due to its close relationship with true dimensional databases used in most OLAP systems, this thesis will consider only dimensional models from now on. The temporal issues discussed, however, can be applied generally.

1.3.4 Information needs

I now describe some of the ways in which the information stored in the data warehouse can be exploited by the directors of the Wine Club in order to assist them with decision making. Although the Wine Club is a fictitious organisation, the classes of information could be applied to almost any trading organisation. It is important that the examples are realistic because, in the next chapter, I will show how some of the query examples that follow may return inaccurate results when variations to attribute values occur over time.

There is quite a wide range of information that could be derived from the dimensional model in the case study. The information may be grouped into several classifications. The following classes should not be considered to be exhaustive.

Member Behaviour

This involves the general tracking of members' purchasing behaviour over time. Additionally, the organisation needs to establish whether the membership is growing or receding, as follows:

- Members whose behaviour is relatively consistent. The club will hope to encourage these members to purchase more wine. Bonus schemes and volume discounts might be offered to members in this category. If half these members could be enticed to increase their purchases by ten percent, it may result in significantly higher profits for the club.

- Members whose purchases are tailing off, both in terms of frequency and quantity. This group of members should perhaps be regarded as in a 'high-risk' category. Their behaviour might suggest that their continued membership is at risk. Information of this kind enables the directors to take positive action to maintain members' interest in the club.

- How many member did we have at the end of last year, compared to the previous year and the year before that? How many members did we recruit over the last quarter compared to the previous quarter? Are there common characteristics about members that have been lost?

Product Line Behaviour

Similar to the tracking of members, this involves the general tracking of product performance over time. In particular, the organisation needs to identify:

- Products whose behaviour is relatively consistent. This information helps the club to identify its 'bedrock' products upon which reasonably reliable forecasts can be made. As with the fairly consistent members, some inducements could be introduced that may help to lift the sales of these products slightly.

- Products whose purchases are tailing off, both in terms of frequency and quantity. It is known that some wines, like many other kinds of products have a 'life cycle'. If the club was able to predict, with reasonable confidence, the likely life cycle of their wines, then the need for expensive 'bin end' sell-offs could be avoided.

House Wine Popularity

One of The Wine Club's biggest sellers is its range of 'House' wines. At any one time, there are two house wines in the catalogue: house red and house white.

The bottles are labelled with the Club's own label. They always carry the same product code. The main requirements of the house wines are that they should be cheap but palatable. The company varies the supplier of the house wines over time depending on the price and quality. It is important that the relative popularity of house wines is monitored so that the club

continues to maximise its revenue from these products. They would like to know, over time, which suppliers have served them best with this range of products.

Area Manager Performance

Each area manager has targets against which their performance is measured. In general, these are:

- Revenue growth targets across the product range.
- Growth of membership numbers in their area

The performance is reviewed quarterly. It would be very useful to be able to monitor performance more frequently so the managers would be able to determine how well their areas were doing instead of having to wait for the quarterly report.

Special Promotions

The company has identified a method of boosting sales by promoting certain events such as:

- The Derby
- The opening of the Trout and Salmon fishing season
- The Admiral's Cup
- The British Formula One Grand Prix
- The British Open Golf Championship
- Last night of the Proms

This would comprise specially labelled wines targeted at specific members. The members' hobby code, together with the type of wines they prefer, could be used in the matching of members and wines to events.

Having reviewed some of the kinds of information that the Wine Club directors need, it will be helpful to show some sample queries that could be executed against the data warehouse to provide at least some of the information needed to answer the questions. These queries are intended to be viewed as examples only. There are usually many different approaches that could be taken. It is important that there is a clear business definition of each problem.

The query in Figure 1.7 summarises Sales quantity and value by season within year. It could be further refined by the addition of the line *'and year = 1998'* into the predicate if there was a requirement to restrict the result set to just a single year.

```
select season,year,sum(quantity) "Bottles Sold",sum(value) "Revenue"
    from time t,sales s
    where s.time_code = t.time_code
    group by season,year
    order by year,season
```

Figure 1.7 Sales quantity and value by season within year

The query in Figure 1.7 and all subsequent Wine Club query examples have been tested against the case study database that has been developed to support the research. Examples of the data held in the database are included in Appendix C.

Figure 1.8 shows a query that summarises Sales quantity and value by season within area. It is really just a refinement of the previous query shown in Figure 1.7 but, by splitting the results into areas, it provides some of the information needed to provide area managers' measurement statistics. It could be further refined to include the manager's details (see further queries below) and to further split the results by wine category by the inclusion of the Manager and Wine tables.

```
select season,year,sales_area_name,
                sum(quantity) "Bottles Sold",sum(value) "Revenue"
    from time t,sales s, member m,sales_area a
    where s.time_code = t.time_code
    and a.sales_area_code = m.sales_area_code
    and m.member_code=s.member_code
    group by season,year,sales_area_name
    order by year,season,sales_area_name
```

Figure 1.8 Seasonal sales revenue by area

Figure 1.9 is a query that summarises sales by Area Manager by Year. It will be used to assess the manager's performance:

```
select c.manager_name,y.year,sum(s.value)
    from manager m, sales_area a,member c,sales s, time t
    where m.manager_code=a.manager_code
    and c.sales_area_code=a.sales_area_code
    and c.member_code=s.member_code
    and s.time_code = t.time_code
    group by m.manager_name, y.year
    order by m.manager_name, y.year
```

Figure 1.9 Sales revenue by Area Manager by year

Another method of assessing the manager's performance is to measure the new members attracted into the club within the manager's sales area. Figure 1.10 shows a query that counts new members by Area Manager by year. It does this by grouping the members by the year in which they joined. It could easily be amended to summarise by quarter, or month, instead of year. Also, the year, or any other period, could be further constrained by the addition of the line *'and year = 1998'* to the predicate.

```
select m.manager_name, t.year, count(*)
    from manager m, sales_area a,member c, time t
    where m.manager_code=a.manager_code
    and c.sales_area_code=a.sales_area_code
    and c.date_joined=t.time_code
    group by m.manager_name, t.year
    order by m.manager_name, t.year
```

Figure 1.10 New members by Area Manager by year

One of the requirements was to be able to determine which members' sales were decreasing. There are many ways of achieving this. Different analysts would have differing views as to how this problem should be tackled. Figure 1.11 shows one way of answering the question. This query lists members whose sales for 1998 are less than 90% of their sales for 1997. It allows for a ten percent drop in sales but reports any decline in sales that is greater. It is achieved by the use of a sub-query.

```
select c1.member_code,c1.member_name,sum(s1.value)
    from member c1, sales s1, time t1
    where c1.member_code=s1.member_code
    and s1.time_code = t1.time_code
    and t1.year = 1998
    group by c1.member_code,c1.member_name
    having sum(s1.value) * 0.90 < (select sum(s2.value)
        from member c2, sales s2, time t2
        where c2.member_code=s2.member_code
        and c2.member_code=c1.member_code
        and s2.time_code = t2.time_codeand t2.year = 1997)
```

Figure 1.11 Members whose purchases for 1998 less than 90% of 1997

There is another way of looking at members who appear to be slightly hesitant in their ordering behaviour. Figure 1.12 shows a query that lists members whose last order was more than sixty days ago.

```
select c.member_name
    from member c
    where c.member_code in(select s2.member_code
    from sales s2
    group by s2.member_code
    having max(s2.time_code) < current_date-60)
```

Figure 1.12 Members without recent sales

There is a stated requirement to monitor the sales of house wines by supplier to ensure that the product continues to be a major contribution to the revenue of the club. The final query in the set of business measures for the Wine Club, shown in Figure 1.13, summarises sales of House wines by type of wine (red or white) and by supplier. In view of the fact that house wines change frequently, a special region code of *'hse'* has been set up to identify house wines.

```
select wine_name,supplier_name,sum(s.value)
    from wine w, sales s, supplier u
    where s.wine_code = w.wine_code
    and w.supplier_code=u.supplier_code
    and region = 'hse'
    group by wine_name,supplier_name
```

Figure 1.13 House wine sales by supplier

1.4 Summary

In this chapter, the importance and popularity of data warehousing to businesses has been described. Data warehouses are a special kind of database with requirements that are different to other kinds of databases. Their structure, the way in which they are populated, the type of use to which they are put and need for the proper representation of time are all aspects of data warehouses that set them apart from traditional, operational databases.

Also this chapter introduced the Wine Club case study database that will be used as the main source of examples.

The dimensional conceptual model was introduced in its two forms, the star and the snowflake. Using a snowflake schema for the Wine Club, example queries were expressed to answer some of the business questions posed by the case study example. The role of time has been introduced as has the main hypothesis of this thesis which is that the representation of time in data warehouses is generally not adequate and this can lead to inaccuracy in the results obtained from queries.

The following points summarise the contents of the remainder of the thesis:

1. What are the implications of time in data warehousing? Looking ahead, I will show that time is often mis-represented in data warehouses and will examine how the mis-representation of time affects the accuracy of information obtained from the warehouse. The problems associated with time are explored in detail in chapter two.

2. How can data warehouse *information* requirements be captured in a way that allows for the corresponding *temporal* requirements to be recorded? Chapter three evaluates eight candidate conceptual data modelling methodologies that have been designed to enable the representation of time. The methods are assessed to establish whether or not they are appropriate for use in a data warehousing context.

3. What implementation techniques are available today that seek to enable time to be accurately represented? In chapter four the thesis examines the implementation methods that have been adopted by practitioners for the representation of time in data warehouses and assesses them for strengths and weaknesses.

4. What can be done to improve the representation of time in data warehouses? In chapters five and six the thesis puts forward a theory as to how the problems relating to time can be resolved and shows how the theory can be implemented in practice. The solutions build on the work of others but, also, some new ideas are introduced and some existing practices are challenged. Finally, an approach to assisting users in estimating the level of accuracy of their data warehouse queries is proposed.

Chapter seven contains a conclusion that reviews the research undertaken and makes suggestions as to possible future avenues of research.

The appendix contains a guide to assist practitioners in their approach to the capture of information and temporal requirements. It also describes a method for helping practitioners to decide how to implement the requirements.

2 The implications of time in data warehousing

2.1 Introduction

As the principal subject of this thesis is the representation of time in data warehouses, this chapter introduces the characteristics of time and the way that it is used in data warehousing applications. The chapter goes on to describe fully the problems encountered with time.

The presence of and the dependence upon time is one of the things that sets data warehouse applications apart from traditional operational systems. Most business applications are suited to operating in the present environment where time does not require special treatment. In many cases, dates are no more than descriptive attributes. In a data warehouse, time affects the very structure of the system. The temporal requirements of a data warehouse are very different to those of an operational system yet it is the operational system that feeds information about changed data to the data warehouse. In a temporal database management system, support for time would be implicit within the DBMS and the query language would contain time specific functions to simplify the manipulation of time. Until such systems are generally available, the data warehouse database has to be designed to take account of time. The support for time has to be explicitly built into the table structures and the queries.

The purpose of time in data warehousing is that it enables historical data to be held and queried upon. This means that users of data warehouses can view aspects of their enterprise at any specific point or over any period of time for which the historical data is recorded. This enables the observation of patterns of behaviour over time so that we can make comparisons between similar or dissimilar periods e.g. this year vs. last year, seasonal trends. Armed with this information, we can extrapolate with the use of predictive models to assist us with planning and forecasting. We are, in effect, using the past to attempt to predict the future:

> ***'If men could learn from history, what lessons it might teach us!***
> ***But passion and party blind our eyes, and***
> ***the light which experience gives is a lantern on the stern,***
> ***which shines only on the waves behind us!'*** (Coleridge (1835))

Despite the gloomy warning from the nineteenth century, the use of information from past events and trends is commonplace in economic forecasting, social trend forecasting and even weather forecasting.

The value and importance of historical data is generally recognised. Youngworth (1995) sees the ability to store historical data as the main advantage of data warehousing as it ensures that no information is hidden from the users and Zagelow and Bontempo (1998) cite the absence of historical data, in operational systems, as one of the motivating factors in the development of data warehouses. Surveyer (1998) says that historical trend and patterns are vital for insights and building predictive models.

Some authors such as Inmon (1992) suggest that most operational systems do keep a limited amount of history, about 60-90 days. In fact, this is not usually the case because the data held at any one time in, say, an order processing system will be orders whose life cycle has not completed to the extent that the order can be removed from the system. This means that it may take, on average, sixty to ninety days for an order to pass through all its states from inception to deletion. Therefore, at any one time, some of the orders may be up to ninety days old with a status of 'invoiced', while others will be younger with different states such as 'new', 'delivered' or 'back ordered' etc. This is not the same as history in our sense. We need to be clear what we mean by history in the context of data.

2.2 The fact data

In a dimensional data warehouse, the source systems from which the facts are derived are the organisation's operational systems such as order processing. The source systems are not usually designed to record or report upon historical information. For instance, in an order processing system, once an order has satisfactorily passed through its life cycle, it tends to be removed from the system by some archival or deletion process. After this, for all practical purposes, the order will not be visible. In any case, it will have passed beyond the state that would make it eligible to be captured for information purposes.

The task of the data warehouse manager is to capture the appropriate entities when they achieve the state that renders them eligible to be entered into the data warehouse. That is, when the appropriate event occurs, a snapshot of the entity is recorded. This is likely to be

before they reach the end of their life cycle. For instance, an order is captured into the data warehouse when the order achieves a state of, say, 'invoiced'. At this point the invoice becomes a 'fact' in the data warehouse. Having been captured from the operational systems, the facts are usually inserted into the fact table using the bulk insertion facility that is available with most database management systems. Once loaded, the fact data is not usually subject to change at all. The recording of history in the fact table is achieved by the continual insertion of such records over time.

Usually each fact is associated with a single time attribute that records the time the event occurred. The time attribute of the event would, ideally, be the 'valid time' i.e. when the event occurred in the real world. In practice, valid times are not always available and transaction times have to be used.

The actual data type used to record the time of the event will vary from one application to another depending on how precise the time has to be (the granularity of time might be day, month and year when recording sales of wine, but would need to be more precise in the case of telephone calls and would probably include hours, minutes and seconds).

2.3 Dimensional data

Operational data, from which the facts are derived, is accompanied by supporting data, often referred to as reference data. The reference data relates to entities such as customers, products and sales regions etc. Its primary purpose, within the operation processing systems, is to enable, for instance, the right products and documentation to be sent to the right addresses. It is this data that is used to populate the dimensions and the dimensional hierarchies in the data warehouse.

In the same way that business transactions have a life cycle, these reference entities also have a life cycle. The life cycle of reference data entities is somewhat different to transactions. Whereas business transactions, under normal circumstances, have a predefined life cycle that starts at inception and proceeds through a logical path to deletion, the life cycle of reference data can be much less clear. The existence of some entities can be discontinuous. This is particularly true of customer entities who may switch from one supplier to another and back again over time. It is also true of some products, such as seasonal products. ' Also, the

attributes are subject to change. For instance, in the UK, approximately ten percent of households change their address each year (Green et al (1997)).

2.3.1 The affect of time on the data model

Organisations wishing to build a data warehouse have often already built a data model describing their operational business data. This model is sometimes referred to as the corporate data model. Kimball refers to it as 'the big chart on the wall of the IS database designer's cubicle'.

When building a data warehouse, practitioners often encounter the requirement to utilise the customer's corporate data model as the foundation of the warehouse model. The organisation has invested considerably in the development of the model and any new application is expected to use it as the basis for development. The original motivation for the database approach was that data should be entered only once and that it should be shared by any users who were authorised to have access to it.

Figure 2.1 depicts a simple fragment of a data model for an operational system. Although the Wine Club data model could be used, the model in figure 2.1 provides a clearer example:

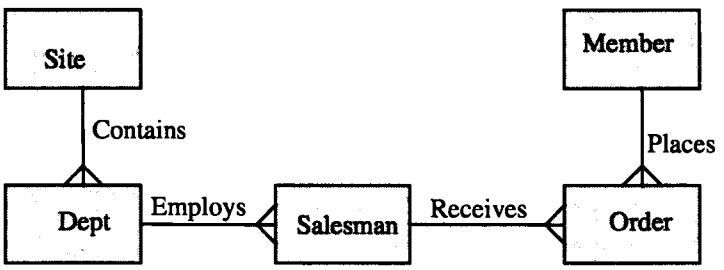


Figure 2.1 Fragment of operational data model

This example is typical of most operational systems in that it contains very little historical data. If we are to introduce a data warehouse into the existing data model, we might consider doing so by introduction of a time variant table that contained the history that is needed.

Taking the above fragment of a corporate data model as a starting point, and assuming that the warehouse subject area is 'Sales', a dimensional warehouse might be created as in Figure 2.2.

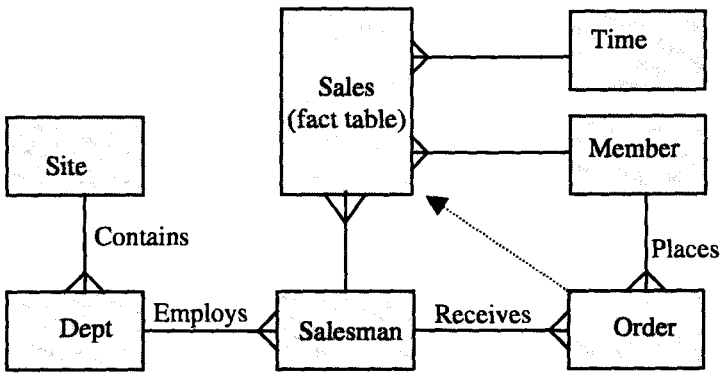


Figure 2.2 Operational model with additional sales fact table

Figure 2.2 shows a dimensional model with the fact table (Sales) at the centre and three dimensions of analysis. These are time, member and salesman. The salesman dimension participates in a dimensional hierarchy in which a department employs salesmen and a site contains many departments. Figure 2.2 further shows that the sales fact table is populated by the data contained in the orders table, as indicated by the dotted arrow (not part of standard notation). That is, all new orders that have achieved the state required to enable them to be classified as sales, are inserted into the sales table and are appended to the data already contained in the table. In this way the history of sales can be built.

At first sight, this appears to be a satisfactory incorporation of a dimensional data warehouse into an existing data model. Upon closer inspection, however, we find that the introduction of the fact table 'Sales' has had interesting effects.

To explain the effect, the salesmans' dimensional hierarchy is extracted as an example, shown in Figure 2.3. This hierarchy shows that a site may contain many departments and a department may employ many salesmen. This sort of hierarchy is typical of many such hierarchies that exist in all organisations.

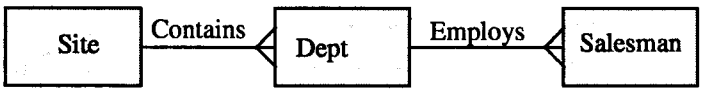


Figure 2.3 Sales hierarchy

The relationships shown here imply that a salesman is employed by one department and that a department is contained in one site. These relationships hold at any particular point in time.

The addition of a fact table, which contains history, is attached to the hierarchy as shown in figure 2.4:

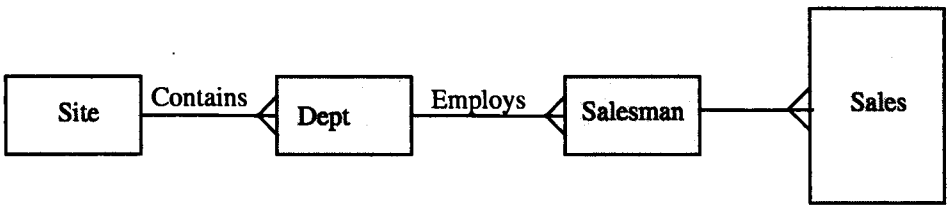


Figure 2.4 Sales hierarchy with sales table attached

The model now looks like a dimensional model with a fact table (sales) and a single dimension (salesman). The salesman dimension participates in a dimensional hierarchy involving departments and sites.

Assuming that it is possible, during the course of ordinary business, for a salesman to move from one department to another, or for a department to move from one site to another, then the cardinality (degree) of the relationships 'Contains' and 'Employs' no longer hold. The hierarchy, consisting of salesmen, departments and sites contains only the latest view of the relationships. Because sales are recorded over time, some of the sales made by a particular salesman may have occurred when the salesman was employed by a different department.

Whereas the model shows that a salesman may be employed by exactly one department, this is only true where the relationship is viewed as a 'snapshot' relationship. A more accurate description is that a salesman is employed by exactly one department *at a time*. Over time, a salesman may be employed by one or more departments. Similarly, a department is contained by exactly one site at a time. If it is possible for departments to move from one site to another

then, over time, a department may be contained by one or more sites. The introduction of *time variance*, which is one of the properties of a data warehouse, has altered the degree of the relationships within the hierarchy and they should now be depicted as many-to-many relationships as shown in Figure 2.5.

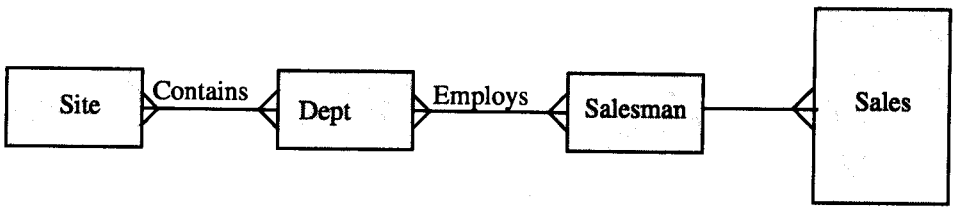


Figure 2.5 Sales hierarchy showing altered relationships

This leads to the following observation:

The introduction of a time variant entity into a time invariant model potentially alters the degree of one or more of the relationships in the model.

Simon (1998e) makes the point that it is the rules of the business, not a technical phenomenon, that causes these changes to the model. The degree to which this causes a problem will vary from application to application. Kimball (1996a) confirms that dimensions typically contain one or more natural hierarchies and my experience tends to support this view. It seems reasonable to assume, therefore, that every organisation intending to develop a data warehouse will have to deal with the problem of the degree of relationships being altered as a result of the introduction of time.

The above example describes the kind of problem that can occur in relationships that are able to change over time. In effect the cardinality (degree) of the relationship has changed from 'one-to-many' to 'many-to-many' due to the introduction of time variance. In order to capture the altered cardinality of the relationships, intersection entities would normally be introduced as shown in figure 2.6

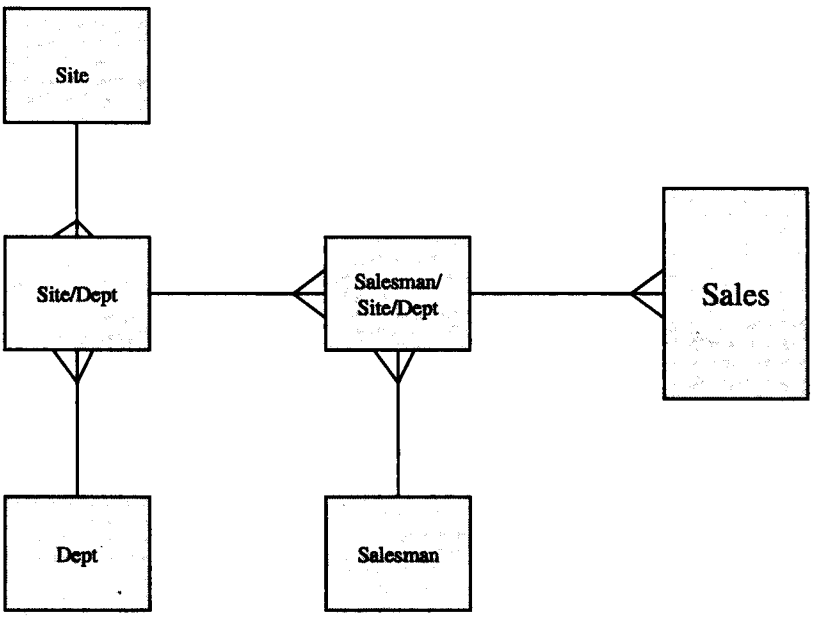


Figure 2.6 Sales hierarchy with intersection entities

This brief introduction to the problem shows that it is not really possible to combine a time-variant data warehouse model with a non time-variant operational model without some disruption to the original model. If we compare the altered data model to the original model, it is clear that the introduction of the time variant sales entity has had some repercussions and has forced some changes to be made. This is one of the main reasons that forces data warehouses to be built separately from operational systems. Some practitioners, such as Paulin (1998), believe that the separation of the two is merely a performance issue in that most database products are not able to be optimised to support the highly disparate nature of operational versus decision support type of queries. This is not the case. The example shows that the structure of the data is actually incompatible. In the future it is likely that operational systems will be built with more 'decision support awareness', but any attempt to integrate decision support systems into traditional operational systems will not be successful.

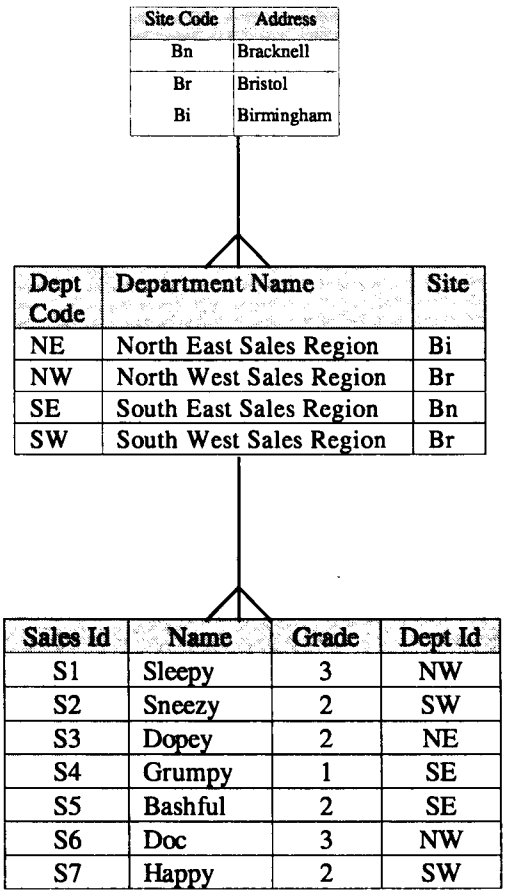
2.3.2 The affect of time on query results

As these entities change over time, in operational processing systems, the new values tend to replace existing values. This gives the impression that the old, now replaced, value never existed. For instance, in the Wine Club example, if a member moves from one address to another and, at the same time, switches to a new region, there is no reason within the order

processing system to record the previous address as, in order to service orders, the new address is all that is required. It could be argued that to keep information about the old address is potentially confusing, with the risk that orders may be inadvertently dispatched to the wrong address.

In a temporal system such as a data warehouse, which is required to record and report upon history faithfully, it may be very important to be able to distinguish between the orders placed by the customer while resident at the first address from the orders placed since moving to the new address. An example of where this information would be needed is where regional sales were measured by the organisation. In the example described above, the fact that the customer, when moving, switched regions is important. The orders placed by the customer while they were at the previous address need to have that link preserved so that the previous region continues to receive the credit for those orders. Similarly, the new region should receive credit for any subsequent orders placed by the customer during their period of residence at the new address. The warehouse needs to record not only the fact that the data has changed but also when the change occurred. There is a conflict between the system supplying the data, which is not temporal, and the receiving system, that is. The practical problems surrounding this issue are dealt with in detail in section 2.5.2.

Later on in this chapter, the consequences of the problem are explored in detail by use of actual data in the case study database. Figure 2.7 provides a simple illustration of the problem by building on the example given. I will start by adding some data to the entities.



Site Code	Address
Bn	Bracknell
Br	Bristol
Bi	Birmingham

Dept Code	Department Name	Site
NE	North East Sales Region	Bi
NW	North West Sales Region	Br
SE	South East Sales Region	Bn
SW	South West Sales Region	Br

Sales Id	Name	Grade	Dept Id
S1	Sleepy	3	NW
S2	Sneezy	2	SW
S3	Dopey	2	NE
S4	Grumpy	1	SE
S5	Bashful	2	SE
S6	Doc	3	NW
S7	Happy	2	SW

Figure 2.7 Sales hierarchy with data

The example in Figure 2.7 shows a 'Relational' style of implementation where the relationships are implemented using foreign key columns. In the data warehouse, the Sales Person dimension would be related directly to the sales fact table. Each Sales fact would include a foreign key attribute that would contain the sales identifier of the sales person who was responsible for the sale.

In order to focus on the impact of changes to these relationships, time is omitted from the following set of illustrative queries.

In order to determine the value of sales by sales person, the SQL query shown in Figure 2.8 could be written:

```
Select name, sum(sales)
    from sales s1, sales_person s2
    where s1.sales_id = s2.sales_id
    group by name
```

Figure 2.8 Total sales by salesman

In order to determine the value of sales by department, the SQL query shown in Figure 2.9 could be written:

```
Select department_name, sum(sales)
    from sales s1, sales_person s2, department d
    where s1.sales_id = s2.sales_id
    and s2.dept_id = d.dept_code
    group by department_name
```

Figure 2.9 Total sales by department

If the requirement was to obtain the value of sales attributable to each site, then the query in Figure 2.10 could be used:

```
Select address, sum(sales)
    from sales s1, sales_person s2, department d, site s3
    where s1.sales_id = s2.sales_id
    and s2.dept_id = d.dept_id
    and d.site = s3.site_code
    group by address
```

Figure 2.10 Total sales by site

The result sets from these queries would contain the sum of the sales value grouped by sales person, department and site.

The results will always be accurate so long as there are no changes in the relationships between the entities. However, as previously shown, changes in the dimensions are quite common.

As an example, if Sneezy were to transfer from department 'SW' to department 'NW', his relationship between the sales person entity and the department entity will have changed. If the same three queries are executed again, the results will be altered.

The results of the first query in Figure 2.8, which is at sales person level, will be the same as before because the sales made by Sneezy are still attributed to him.

However, in Figure 2.9, which is at the department level, all sales that Sneezy was responsible for when he worked in department 'SW' will in future be attributed to department 'NW'. This is clearly an invalid result.

The result from the query in Figure 2.10, which groups by site address, will still be valid because, although Sneezy moved from SW department to NW department, both SW and NW reside at the same address, Bristol. If Sneezy had moved from SW to SE or NE, then the Figure 2.10 results would be incorrect as well.

The example so far has focused on how time alters the cardinality of relationships. There is, equally, an affect on some attributes. If we look back at the Sales Person entity in the example, there is an attribute called 'Grade'. This is meant to represent the sales grade of the sales person. If we want to measure the performance of sales people by comparing volume of sales against grades, this could be achieved by the query in Figure 2.11:

```
Select grade, sum(sales)
    from sales s1, sales_person sp1
    where s1.sales_id = sp1.sales_id
    group by grade
```

Figure 2.11 Total sales by salesman's grade

If any sales person has changed their grade during the period covered by the query, then the results will be inaccurate because all their sales will be recorded against their current grade. In order to produce an accurate result, the periods of validity of the sales person's grades must be kept. This might be achieved by the introduction of another intersection entity.

I will now use the Wine Club case study data warehouse to illustrate the point and give us some baseline query results so that, in a later chapter, some of the candidate solutions can be tested.

The baseline query to be used is now shown. Its purpose is to summarise Sales quantity and value by Area. The SQL required to execute this query is as shown in Figure 2.12:

```
select sales_area_name "Sales Area",
                    sum(quantity) "Bottles",sum(value) "Revenue"
  from sales s, member c,sales_area a
  where a.sales_area_code=c.sales_area_code
  and c.member_code=s.member_code
  group by sales_area_name
  order by sales_area_name
```

Figure 2.12 Total quantity and revenue by sales area query

The query in Figure 2.12 produces the following result set.

Sales Area	Bottles	Revenue
North East	9628	65083.05
North West	4077	29329.06
South East	5662	39597.12
South West	5907	41560.58

Figure 2.13 Total quantity and revenue by sales area result set

This result set shown in Figure 2.13 can be referred to as the base result set. That is, it should be regarded as the result set representing the *true* answer to the query, before any changes have been made.

Figure 2.14 illustrates the residential moves of three members, from the south eastern area, to the south western area using the following set of statements. Each update statement changes two attributes. The first effects the change of address and the second is necessary because the change in addresses has created a need to change the sales area codes (see Appendix C for the original member details).

```
Update member
    set member_address='44 Sea View Terrace, West Bay, Bridport, Dorset',
    sales_area_code='SW'
    where member_code = 1136

Update member
    set member_address='Bay View Cottage, The Promenade, Weymouth, Dorset',
    sales_area_code='SW'
    where member_code = 1651

Update member
    set member_address='Flat 10, The Heights, Cherry Road, Torbay, Devon',
    sales_area_code='SW'
    where member_code = 1404
```

Figure 2.14 Queries to implement residential moves

If the query shown in Figure 2.12 is re-executed against the altered tables, the results shown in Figure 2.15 are produced.

Sales Area	Bottles	Revenue
North East	9628	65083.05
North West	4077	29329.06
South East	4261	29975.03
South West	7308	51182.67

Figure 2.15 Revised total quantity and revenue by sales area result set

Comparing these results to the base set shown in Figure 2.13, it is obvious that there are differences. All of the sales previously made to the three members in question have now been attributed to sales area 'SW' even though none of the sales to these three members actually occurred when they were resident in the SW region.

The database is now returning inaccurate results. Whether the level of inaccuracy is acceptable is a matter for the directors of the organisation to decide. Over time, however, the

information would become less and less accurate and the value of the information is likely to become increasingly questionable.

The reason the results have changed is equally obvious. Three rows of the set of members have been altered. In real-world terms the changes were due to the members having changed their addresses to new addresses in a different sales area. This would be a common change in most data warehouse applications.

The effect on the database is quite profound in that all the sales of wine made to those members are now attributed to their new sales area even though the actual sales were made in their original sales area. The original 'Sales_Area_Code' attribute has simply been overwritten.

It is important to note that the same problem exists with the addresses. It was stated earlier that there is a requirement to scan members' addresses to enable a finer grain of query to be executed on addresses. If a search was made on towns that included any of 'Bridport', 'Weymouth', or 'Torbay' the results would be inaccurate. Also, any search on the towns from which these members had moved would also return incorrect results.

Returning to the seven example queries that were shown in Figures 1.7-1.13, we can assess the likely impact of changes over time of each query. The first query is 'Sales quantity and revenue by season within year'. There is only one dimension used in this query and that is the time dimension. The attributes of the time dimension are not subject to change over time and so this query will continue to work. The query in Figure 1.8 is 'Seasonal sales revenue by area'. This query uses the sales area dimension. As we have seen, the sales area is highly likely to be affected over time as members move from one area to another. With the passage of time, therefore, the results returned by this query will become more and more inaccurate. The queries in Figures 1.9 and 1.10 also use the sales area dimension. Additionally, they both use the manager dimension. The accuracy of these queries will be affected by changes in manager as well as movements of members between sales areas. The queries in Figures 1.11 and 1.12 use the member dimension. Changes to member attributes used in the query are unlikely to affect the accuracy of these two queries since the only attribute that can change is the member's name and the club is likely to be interested only in the members latest name. The

final query (Figure 1.13) uses the wine and supplier dimensions. Over time, if the Wine Club switches from one supplier to another, the query will return inaccurate results.

So out of the seven queries initially specified, four will become increasingly less accurate over time due to changes that occur during the natural course of business.

2.3.3 The time dimension

The time dimension is a special dimension that contains information about times. For every possible time that may appear in the fact table, an entry must exist in the time dimension table. This time attribute is the primary key to the time dimension. The non key attributes are application specific and provide a method for grouping the discrete time values. The groupings can be anything that is of interest to the organisation. Some examples might be:

- Day of week
- Week end
- Early closing day
- 24 hour opening day
- Weather conditions
- Week of year
- Month name
- Financial month
- Financial quarter
- Financial year

Some of the groupings, listed above, could be derived from date manipulation functions supplied by the database management system whereas others, clearly, cannot.

2.4 The effect of causal changes to data

Upon examination, I have found that some changes are causal in nature, in that a change to the value of one attribute implies a change to the value of some other attribute in the schema. The extent of causality will vary from case to case but the designer must be aware that a change to the value of a particular attribute, whose historical values have low importance to the organisation, may cause a change to occur in the value of another attribute that has much greater importance. Whilst this may be true in all systems, it is particularly relevant to data warehousing because of the disparate nature of the source systems that provide the data used to populate the warehouse. It is possible, for instance, that the source system containing customer addresses may not actually hold information about sales areas. The sales area classification may come from, say, a marketing database or some kind of demographic data. Changes to addresses, which are detected in the operational database, must be implemented at exactly the same time as the change to the sales area codes.

Acknowledgement of the causal relationship between attributes is essential if accuracy and integrity is to be maintained. In the logical model it is necessary to identify the dependencies between attributes so that the appropriate physical links can be implemented.

2.5 Capturing Changes

I will now examine how changes are identified in the source systems and can be subsequently captured into the data warehouse and the problems that can occur.

2.5.1 Capturing facts

As has been previously stated, the facts relate to the business transactions of the organisation. Facts are usually derived from some entity having been 'frozen' and captured at a particular status in its life cycle. The process by which this status is achieved is normally triggered by an event. There are some, such as Snodgrass (1997), who consider the facts and the events to be synonymous and state that facts *are* events. The definition of an event from the glossary of temporal database concepts (Jensen et al 1994) is that it is 'an instantaneous fact' that occurs at a single point in time. This means that there is only one time attribute associated with a fact and that is the time of its occurrence.

'An event is such a little piece of time and space,
you can mail it through the slotted eye of a cat'

(Diane Ackerman (1991) page 167)

There are two ways of considering the definition of an event. If the data warehouse is viewed in isolation so that the facts that it records are not perceived as related to the source systems from which they were derived, then they can be viewed purely as events that occurred at a single point in time. If, however, the data warehouse is perceived as part of the 'enterprise' database systems, then the facts should be viewed within the wider context and they become an entity preserved at a 'frozen' state, having been triggered by an event. Either way, the distinguishing feature of facts is that they do not have a life span. They are associated with just one time attribute. For the purpose of clarity, in this thesis, the following definition of facts will be adopted:

A fact is a single state entity
that is created by the occurrence of some event.

In principle, the processes involved in the capture of fact data are relatively straightforward. The extraction of new facts for insertion into the data warehouse is performed on a periodic, very often daily, basis. This tends to occur during the time when the operational processing systems are not functioning. Typically this means during the overnight 'batch' processing cycle. The main benefit of this approach is that all of the previous day's data can be collected and transferred at one time. The process of identifying the facts varies from one organisation to another and can vary between being very easy to almost impossible to accomplish. For instance, the fact data may come from:

- Telephonic network switches or billing systems, in the case of telecommunications companies

- Order processing systems as in the case of mail order companies such as the Wine Club.

- Till receipts in the case of retail outlets

Once the facts have been identified they are usually stored into sequential files or streams that are appended-to during the day. As the data warehouse is usually resident on a hardware

platform that is separate from the operational system, the files have to be moved before they can be processed further.

The next step is to validate and modify each record to ensure that it conforms to the format and semantic integration rules that were described in chapter 1.

The actual loading of the data is usually performed using the 'bulk' load utility that most database management systems provide.

Once recorded, the values of fact attributes never change so they should be regarded as single state, or stateless. There is a time element that applies to facts but it is simply the time that the event occurred. It is usually implemented in the form of a single timestamp. The timestamp will vary, in granularity, from one application to another. For instance, in the Wine Club, the timestamp of a sale records the date of the sale. In a telecommunications application, the timestamp would record, not only the date but also, the hour, minute and second that the call was placed.

2.5.2 Capturing dimensions

The dimensions are derived from what has been referred to as the reference entities within the organisation. This is information such as customer, product and market segment. Unlike the facts in a data warehouse, this type of information does have a lifespan. For instance, products may have various states during their lifespan from new to fast-moving to slow-moving to discontinued to deleted.

The identification and capture of new or changed dimensional information is usually quite different to the capture of facts. For instance, it is often the case that customer details are captured in the operational systems some time after the customer starts using the services of the organisation. Also, the date at which the customer is enrolled as a customer is, often, not recorded in the system, neither is the date when they cease to become a customer.

Similarly, when a dimensional attribute changes, such as the address of a customer, the new address is duly recorded in such a way as to replace the existing address. The date of the change of address is often not recorded. The dates of changes to other dimensional attributes are also, usually, not recorded. This is only a problem if the attribute concerned is one for

which there is a requirement to record the historic values faithfully. In the Wine Club, for instance, the following attributes need to have their historic values preserved:

- Members' addresses
- Members' sales areas
- Wine sales prices and cost prices
- Suppliers of wines
- Managers of sales areas

If the time of the change is not recorded in the operational systems, then it is impossible to determine the valid time that the change occurred. Where the valid time of a change is not available, then it may be appropriate to try to ascertain the transaction time of the change event. This would be the time that the change was recorded in the database, as opposed to the time the change actually occurred.

However, in the same way that the valid time of changes is not recorded, the transaction time of changes is usually not recorded explicitly as part of the operational application. In order for the data warehouse to capture the time of the changes, there are methods available that can assist us in identifying the transaction times:

1. Make changes to the operational systems. The degree to which this is possible is dependent on a number of factors. If the system has been developed specifically for the organisation, either by an organisation's own IT staff or by some third party, as long as the skills are available and the costs or timescales are not prohibitive, then the operational systems can be changed to accommodate the requirements of the data warehouse. Where the application is a standard package product, it becomes very much more difficult to make changes to the system without violating commercial agreements covering such things as upgrades and maintenance. If the underlying database management system supporting the application is relational, then it is possible to capture the changes by the introduction of such things as database triggers. Experience shows that most organisations are reluctant

to alter operational applications in order to service informational systems requirements for reasons of cost and complexity. Also, the placing of additional processing inside of operational systems is often seen as a threat to the performance of those systems. These points are confirmed by Devlin (1997).

2. Interrogation of audit trail. Some operational applications maintain audit trails to enable changes to be traced. Where these exist, they can be a valuable source of information to enable the capture of transaction time changes.

3. Interrogation of DBMS log files. Most database management systems maintain log files for system recovery purposes. It is possible, if the right skills are available, to interrogate these files to identify changes and their associated transaction times. This practice is discouraged by the DBMS vendors as log files are intended for internal use by the DBMS. If the files are damaged by unauthorised access, the ability of the DBMS to perform a recovery may be compromised. Also, the DBMS vendors always reserve the right to alter the format of the log files without notice. If this happens, processes that have been developed to capture changes may stop working or may produce incorrect results. Obviously, this approach is not available to non-DBMS applications.

4. File comparison. This involves the capture of an entire file, or table, of dimensional data and the copying of the file so that it can be compared to the data already held in the data warehouse. Any changes that are identified can then be applied to the warehouse. The time of the change is taken to be the system time of the detection of the change i.e. the time the file comparison process was executed.

Devlin (1997) goes into more detail on this subject. He divides the incremental capture of changes into two classes, immediate and delayed. The first three of the above solutions are examples of immediate capture in the sense that the changes are captured at the transaction time of the change event. The fourth method is an example of delayed capture.

Experience shows that the file comparison technique is the one most frequently adopted when data warehouses are developed. Devlin confirms this. It is the approach that has least impact on the operational environment and it is the least costly to implement. Unfortunately, it is also the approach that yields the least accurate transaction times in most implementations. Kimball

(1996d) says that the ideal situation is one in which the source system attaches timestamps to the records and isolates the changes so they can be extracted easily. However, he concludes that the most likely solution is, again, a file comparison. In another paper (Kimball (1998d)) he reminds us that some dimensions are created by the amalgamation of data from several operational systems and some external systems. This will certainly exacerbate an already complex problem.

Where the dimensions in a dimensional model are large (some organisations have several million customers), the capture of the data followed by the transfer to the data warehouse environment and subsequent comparison is a process that can be very time consuming. Consequently, most organisations place limits on the frequency with which this process can be executed. At best, the frequency is weekly. The processing can then take place over the weekend when the systems are relatively quiet and the extra processing required to perform this exercise can be absorbed without too much of an impact on other processing. Many organisations permit only monthly updates to the dimensional data and some are even less frequent than that.

The problem is that the only transaction time available, against which the changes can be recorded, is the date upon which the change was discovered (i.e. the file comparison date). So, for example, let us assume that the frequency of comparison is monthly and the changes are captured at the end of the month. If a customer changes address, and geographical region, at the beginning of the month then any facts recorded for the customer during the month will be credited permanently to the old, incorrect region.

It is possible that, during a single month, more than one change will occur to the same attribute. If the data is collected by the file comparison method, the only values that will be captured are those that are in existence at the time of capture. All intermediate changes will be missed completely.

The degree to which this is a problem will vary from application to application. It is accepted that, in general, valid time change capture for dimensions is not, practically speaking, realistic. However, it is important that practitioners recognise the issue and try to keep the difference between transaction time and valid time as small as possible. The fact that some data relating

to time as well as other attributes is found to be absent from the source systems is recognised by practitioners such as Celko and McDonald (1995), Strong et al (1997) and Atre (1998c) as aspects that can come to dominate data warehouse developments. Kimball (1996c and 1996d) describes it as the longest and riskiest part[2].

2.6 The 'Supplies from stock' problem

Another problem with accuracy of reporting, which is related to time, is one occurring in organisations that purchase goods into stock and sell the same goods from that stock. The stock forms a kind of buffer between the supplier and the customer. The problem occurs, in particular, in retail organisations where the retail company, perhaps a supermarket, keeps a stock of products, such as Cox's Apples, for supply to their customers.

As customers purchase the apples, the stock of apples would be slowly depleted. Most of the products have a 're-order' point. This is the stock level which, when reached, triggers a new order to be made to replenish the stock of the product to enable the supermarket to satisfy demand for the next period.

From time to time, the supermarket may switch from one supplier to another. So, for instance, the supplier for Cox's Apples would be changed from the old supplier to the new supplier. Subsequently, this change of relationship would be implemented in the data warehouse application and would take effect from then on.

The data now reflects the reality of the situation as the new supplier is indeed the actual supplier of Cox's Apples. The problem is that the supermarket would still hold stocks of Cox's Apples from the previous supplier. Any sales of these apples, which will be recorded in the data warehouse, will be assumed to be supplied by the new supplier. For a period of time, the supermarket may be supplying apples from both suppliers, even in the same sale. A customer will not be able to tell that their bag of apples came from different suppliers and neither will anyone writing queries against the data warehouse.

---

[2] The quality of data extracted from source systems is a major problem in data warehouse applications and is described in detail by Kimball (1996e). It is not considered further in this thesis.

The Wine Club is a retail organisation of sorts and, potentially, suffers from precisely this problem. There is a requirement in the club to monitor the popularity of wine that may come from different suppliers, such as the house wines, as this might influence its buying policy in the future. At any one time, therefore, the bins may contain wines from any number of suppliers but the latest supplier will be the one assumed to be responsible for the supply on each sale until the next change in supplier.

The problem not only occurs due to changing relationships, it also occurs when attributes change their values. When the vintage of the particular wine changes, the change is not abrupt in the sense that there will, probably, be existing stocks of the wine with its previous vintage. At some point, the Wine Club will find it is purchasing new stocks of wine with the new vintage while it still has some stock of the wine with the previous vintage. The date of the change does not, in itself, have any meaning with respect to constraining queries. It cannot be assumed that, from the date the change was made, all bottles sold will be of the new vintage. This problem is referred to by R. Kimball (1996a) and will be reviewed in the next chapter.

2.7 Summary

In this chapter I have shown that maintaining accuracy in a data warehouse presents a challenging set of problems that are summarised below:

1. Identifying and capturing the temporal requirements. The first problem is to identify the temporal requirements. There is no method to do this currently. The present data warehousing modelling techniques do not provide any real support for this.

2. Capture of dimensional updates. What happens when a relationship changes (e.g. a salesman moves from one department to another) ? What happens when a relationship no longer exists (e.g. a salesman leaves the company) ? How does the warehouse handle changes in attribute values, for example: a product was blue, now it is red? Is there a need to be able to report on its sales performance when it was red or blue, as well as for the product throughout the whole of its life cycle ?

3. The timeliness of capture. It now seems clear that absolute accuracy in a data warehouse is not a practical objective. There is a need to be able to assess the level of inaccuracy so that a degree of confidence can be applied to the results obtained from queries.

4. Synchronisation of changes. When an attribute changes, a mechanism is required for identifying dependent attributes that might also need to be changed. The absence of synchronisation affects the credibility of the results.

I have shown that obtaining the changed data can involve complex processing and may require sophisticated design to implement in a way that provides for both accuracy of information and reasonable performance.

In this chapter I have described the various problems associated with time in data warehousing. Some of these problems are inherent in the standard dimensional model but it is possible to overcome these problems by making changes to the way dimensional models are designed. There are various solutions that might be applied and these are investigated in subsequent chapters of this thesis.

Some of the problems relate to the way data warehouses interact with operational systems. These problems are more difficult to solve and, sometimes, impossible to solve. Nevertheless, data warehouse designers need to be fully aware of the extent of the problems and cogniscent of the various approaches to solving them. These are discussed in the coming chapters and are integrated into the practitioner's guide.

The biggest set of problems lies in the areas of the capture and accurate representation of historical information. The problem is most difficult when changes occur in the lifespan of dimensions and the relationships within dimensional hierarchies. Also where attributes change their values and there is a requirement to faithfully reflect those changes through history.

Having posed the research questions and established the problems I now investigate how the questions can be answered and the problems overcome.

3 General conceptual models for time

3.1 Introduction

The purpose of this chapter is to explore and review the available notations and methods to determine their suitability for use in the production of conceptual models for data warehouses. In view of the temporal nature of data warehouses, the main requirement of a model is that it must be able to properly represent both the information and the temporal requirements of the facts, dimensions, hierarchies and attributes. Aside from the time dimension requirements, there is no formalised approach to gathering temporal requirements available to data warehouse practitioners at the present time. This has led to the temporal requirements being overlooked and, consequently, not satisfied. If practitioners had been encouraged to adopt a methodology that forced them to consider the temporal aspects more fully, then the implemented solutions would undoubtedly be more accurate than they are.

None of the practitioners have a conceptual model for data warehousing even though there are many methodologies. Authors such as Kelly (1996), Brackett (1996), Inmon et al (1997), Dodge and Gorman (1998) and Devlin (1997) all have approaches to data warehouse development but provide no practical direction toward a conceptual model. Examples in the data warehousing literature tend to use the entity relationship approach by default and all reference to time is omitted, except for the time dimension.

The models under review in this chapter, with the exception of the entity relationship model, all claim to provide support for time. The purpose of the review is to determine whether they may be suitable for adoption as data warehouse conceptual models.

In investigating methods for producing conceptual models for data warehouses, there are two other requirements to be considered. The first of these is that the method should provide support for dimensional models. A dimensional model has the property of a kind of radial symmetry that has the facts at the centre with the dimensions radiating outwards from the facts. The facts and the dimensions are quite distinct in the minds of users and developers of data warehouses and this distinction, as well as the radial shape, should be maintained so that readers of the diagram can quickly discern the features of the model. The second additional requirement, which follows on from the first, is that the model should be simple to read and

simple to produce so that non-technical users can produce their own models. A principal requirement that follows from this is that the model must include a diagram. One model that was investigated, called the Temporal Entity Relationship Model (TERM) (Klopprogge (1981) and Klopprogge and Lockemann (1983)), consisted entirely of Pascal-like data structures and functions and did not include any form of diagram. This type of approach is very difficult to read by any but the most technical of people and is entirely unsuitable as a method to be used in conjunction with non-technical business people.

This additional requirement that the model should be comprehensible to business people, as well as providing enough information to enable database designers to develop a logical model, raises the question as to whether a *single* model can be built to satisfy both sets of requirements. This issue will be considered in the review of the models under examination.

The absence of any specialised data warehousing model has led to the default adoption of the Entity Relationship model notation. In view of this, the Entity Relationship approach is reviewed initially in section 3.2 to assess its suitability for use as a model for data warehouses. Although Entity Relationship models do not inherently provide support for time, it is possible to represent time using intersection entities etc. However, the research has shown that there are methodologies that have been developed to provide general support for time for any kind of database and these are examined and reviewed in section 3.3 to assess their suitability for use in data warehousing. The focus of the research remained on data modelling methodologies and this thesis does not claim to have examined all modelling techniques. In the main, the methodologies under review have been developed in parallel with the research into temporal database management systems and are, therefore, designed to be applicable for use by temporal database designers generally. They are mostly based on, or are extensions to, the entity relationship modelling method described by Chen (1976) or Palmer (1978) or the extended entity relationship modelling method described by Elmasri et al (1990). The approach taken for each methodology is as follows: Firstly, the main components of the models are described, with attention being paid to the approach to the representation of time and the description and meaning of the symbols used in the notation. Secondly the Wine Club dimensional model is drawn using the notation of the methodology. In section 3.4, an overall review is conducted that compares the different approaches and discusses the relative

strengths and weaknesses. Finally, section 3.5 analyses what has been learned and what might be carried forward to help formulate a solution.

The specific features of the Wine Club that require temporal support are:

- The dimensions *'Member'*, *'Wine'* and *'Manager'*
- The relationships *'Manages'*, *'Contains'* and *'Supplies'*
- The attributes *'Member.Address'*, *'Wine.Vintage'*, *'Wine.Bottle_Price'*, *'Wine.Bottle_Cost'*, *'Wine.Case_Price'*.

3.2. The Entity Relationship model

Although time variance can be modelled using ER notation, it has no implicit support for time. For example; we know that snapshot relationships, having a cardinality of one-to-many, can be transformed into many-to-many relationships over time. The ER method allows for many to many relationships to remain unresolved in the conceptual model. If explicit resolution of these complex relationships is required, then the transformation must be modelled by the addition of intersection entities in order to enable the time associated with changes in relationships to be represented.

Due to the ubiquitous nature of ER models, modelling tools and generally available ER modelling skills, it is the obvious starting point for an investigation such as this. The first step, therefore, will be to examine the ER method closely so as to determine its suitability for modelling the temporal aspects of dimensional data warehouses.

It is assumed that the reader of this document is familiar with ER models. No explanation of the syntax and symbols is given. For readers requiring to gain more familiarity, attention is drawn to Chen's (1976) paper and published literature such as 'Conceptual Database Design – an entity relationship approach' by Batini et al (1992).

Several nuances of the ER method have evolved including the Integration Definition for Information Modelling (IDEF1X) by Bruce (1992) and Information Engineering(IE) approach

(which is the method used by the Open University). The IE method is the one adopted in this document.

The full ER diagram, for the Wine Club data warehouse, which has been modified from the original model (Figure 1.6) to allow the m:n relationships to be shown, is as follows:

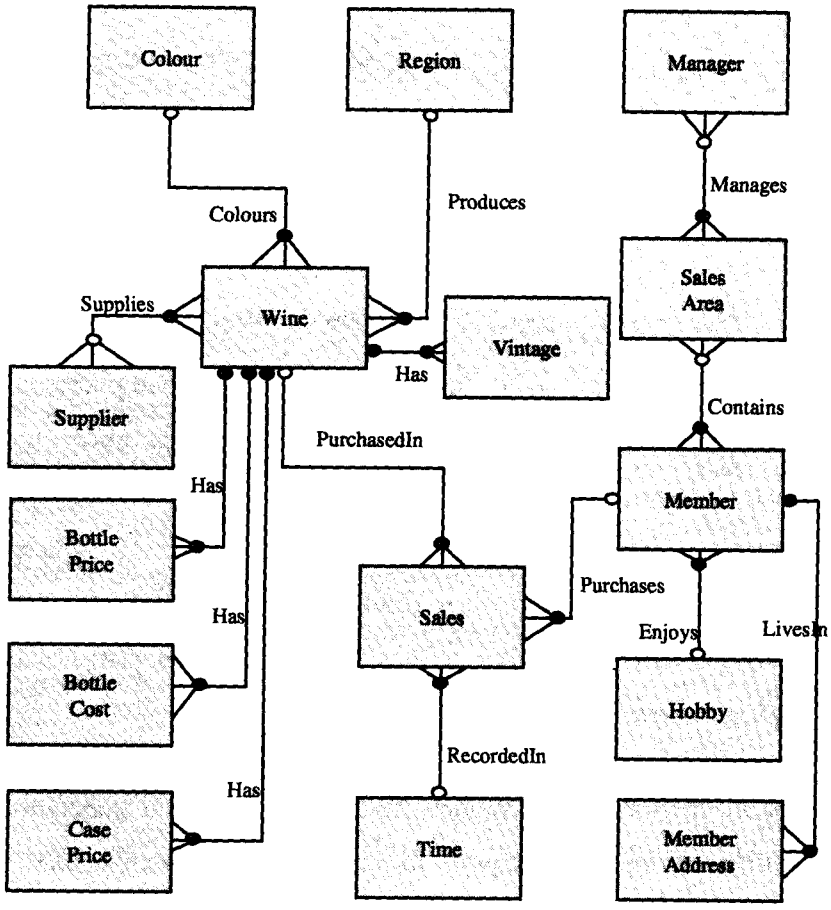


Figure 3.1 Entity relationship dimensional model for the Wine Club

The ER diagram above has been extended to accommodate some of the requirements with respect to time. In order to do this, some of the relationships have been altered from having a cardinality of 1:n to a cardinality of m:n. Additionally, some of the attributes that require temporal support have been transformed into entities which, in turn, have mandatory 1:n relationships with the entities that they were previously attached-to as attributes.

It is not obvious to a designer, when attempting to progress the model, from the conceptual to logical level, that some of the model's components exist purely to accommodate the

requirements for time. Also, the m:n relationships that exist between, for example, the sales area and member entities have arisen for the same reason, that is, to identify the fact that temporal support is required for the relationship. Again, this is not clear from the diagram. The m:n relationships may exist for any reason.

There is another requirement that is not modelled. This relates to the existence of some of the dimensions over time. For instance, the member, wine and manager entities require temporal support for existence and there is no way of describing this within the method. It is not expected that temporal requirements such as this need necessarily be shown on the diagram itself but the methodology should provide some means of recording these requirements somewhere. Currently, if this general method were to be adopted then it would have to be modified to accommodate the requirements.

The model has also lost the essential dimensional shape. The snowflake schema, from which it was derived, is no longer discernible. In order to accommodate the increased number of entities on the diagram and to maintain some degree of readability it is necessary to re-position some the symbols. This inevitably means that the radial nature of the diagram becomes less obvious, almost to the extent that it is no longer distinguishable as a dimensional model. It is important that the fact entity can be easily distinguished from the dimensional entities as it is the fact table that forms the centre of the model and, in effect, gives the diagram its shape. There is no way of classifying entities into different types, using the ER approach. It is true that more effort could have been put into the arrangement of the symbols in an attempt to retain the dimensional appearance. It should be remembered that the dimensional model for the Wine Club is a relatively simple one in having just three dimensions. According to Kimball (1998c), the average dimensional model has between six and ten dimensions so most models will become very complex and difficult to read, especially by non-technical people, as Brackett (1996) says:

> ***'Entity relationship diagrams are usually technically correct but are culturally unacceptable'. (page 112)***

Where, attributes are included on the diagram he says they become confusing when an entity has many data attributes. Kimball (1996a) goes further:

'Entity relation data models are a disaster for querying because they cannot be understood by users and they cannot be navigated usefully by DBMS software. Entity relation models cannot be used as the basis for enterprise data warehouses'.(page 9)

In another article Kimball (1995b) says:

'There is no simple way to change an ER model into a dimensional model, even when you model the same data'.

It is fairly obvious that, in some respects, Kimball is confusing Entity Relationship modelling and normalisation because, in another paper (Kimball (1997a)) he states:

'ER modelling is a logical design technique that seeks to remove redundancy in data'.

Sen and Jacob (1998) also claim that the ER diagrams are not useful in data warehousing because data warehouses are concerned with business structure rather than being aligned to an application. This displays a lack of understanding about the ER modelling method as does the comment by Pasahow (1997) stating that tools that work well for data modelling on operational systems may actually cause harm on data warehousing projects. The statement is justified on the principle that:

'Resolving issues associated with data models requires long meetings that are difficult to convene'.

By this it is assumed he means that ER models are difficult for non-technical people to understand easily without the assistance of trained people and that the diagram is complex in the sense that users have to understand the notation in order to be able to construct or read it. It is not reasonable to expect non-technical people to be able to understand these diagrams in the same way that practitioners, who have received the proper training, would be able to.

With regard to support for time, while the Wine Club example in Figure 3.1 shows it is possible to include time in an ER model, such support is not inherent. Changing the cardinality of relationships from 1:n to m:n does not necessarily convey the message that temporal support is included. This approach is used in ER models for all many to many relationships. It is only when the relationship is resolved by the insertion of an intersection

entity, together with its entity description, that the need for temporal support could be deduced. The introduction of intersection entities causes the diagram to become more complex and exacerbates the problem of ease of understanding.

The conclusion drawn from this is that the general ER approach does not provide an obvious solution to the requirements of data modelling for dimensional data warehouses. There now follows an examination of some of the models that have specialised their treatment of time.

3.3 The Relationships, Attributes, Keys and Entities (RAKE) Method

The Relationships, Attributes Keys and entities Model (RAKE) technique, presented by Ferg (1985) was originally developed in 1984 and was originally designed for use on a US Federal Reserve Board project involving Banking Statistics.

The Banking statistics project (STAT) was a major software development. Its objective was to

> ***'Store data on the history, attributes and inter-relationships of American and foreign financial institutions'.***

RAKE is a variation of Chen's (1976) design for Entity-Relationship modelling. Some of the constructs and conventions are altered and new ones introduced but most of the original semantics have been left intact.

Entities are still represented by rectangles. However all entities now have a keybox in the top left hand corner as the example in Figure 3.2 shows.

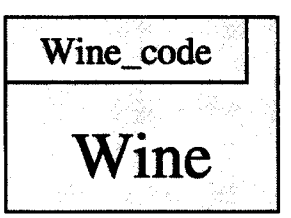


Figure 3.2 RAKE entity

Where an entity cannot be uniquely identified by its own attributes, there is a relationship, either implicit or explicit, with another entity such that the subordinate entity's identifier is concatenated to the superior entity's identifier to provide uniqueness. Where this occurs it is

represented in RAKE by placing the superior part of the identifier on top of the keybox, outside of the entity rectangle as shown in Figure 3.3.

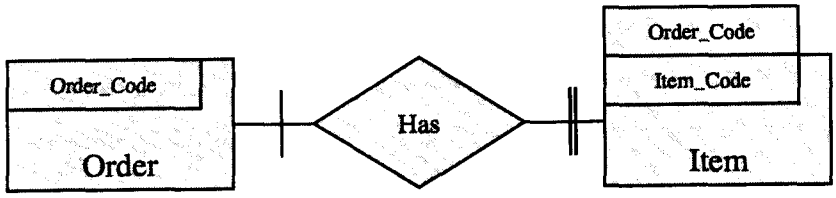


Figure 3.3: 1:n relationship using RAKE

According to the author, this diagramming convention is designed to convey, as graphically as possible, the fact that the identifier for 'Item' is the concatenation of the Order_Code and the Item_Code.

As can be seen from the above example, relationships are depicted using diamonds, in the usual Chen notation.

Non-identifying attributes are represented by circles containing the name of the attribute. The attribute circles are attached to the entity. The actual display of the attribute circles is not strictly enforced. The reason for this is not clear but it might be assumed that the purpose is to reduce the clutter on the diagram.

There are four time-related components in RAKE. These are:

- Events
- States
- Time-Points
- Time-Periods

The author describes history as consisting of a series of *states* succeeding one another in time. This series is punctuated by *events* that transform one state into the next. 'States are described as having duration, whereas events have no duration but occur instantaneously at a single point

in time. A state exists for a *time-period* that extends from its originating *time-point* until its ending time-point. Time-points are occurrences of a data type called *Tstamp*. A Tstamp is a date/time stamp, the granularity of which is determined by whatever is appropriate for the system. The pair of Tstamps that define a time-period are called the BEGINstamp and ENDstamp.

In RAKE, there is no explicit process for identifying which attributes and relationships require time support and which do not.

RAKE uses the Chen Entity-Relationship notation to model relationships. The relationship shown in Figure 3.4 is one in which the degree of 'one-to-many' applies at any point in time but that changes to 'many-to-many' when time is introduced.:

So:

Becomes:

Figure 3.4 Transformation of a temporal relationship in RAKE

The introduction of history further transforms the relationship from binary to ternary as is shown in Figure 3.5.

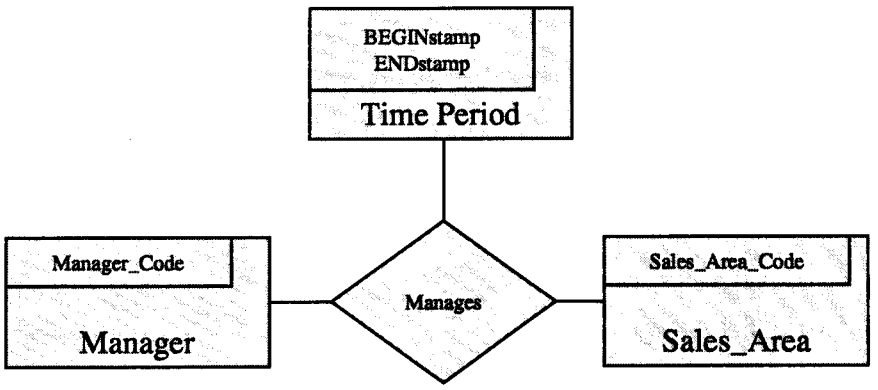


Figure 3.5 Ternary representation of a temporal relationship in RAKE

To avoid cluttering the diagram with Time-Period rectangles, RAKE introduces an equivalent notation in which temporal relationships are represented as weak entity types and this is shown in Figure 3.6. This is equivalent to the introduction of an intersection entity.

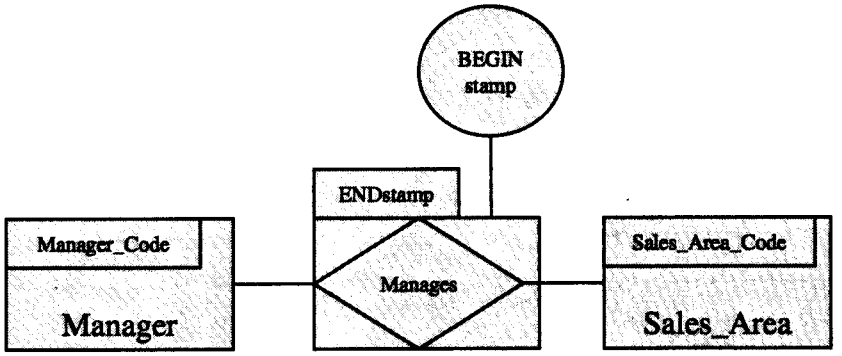


Figure 3.6 Shorthand ternary representation of a temporal relationship in RAKE

The relationship has been replaced by a temporal entity where the primary key consists of the combination of the participating entities and the End timestamp. The notation allows for the keybox to contain just the additional attribute. The inclusion of the primary keys of the participating entities, and the fact that the relationship between them is now m:n, is implicit.

So the 'Manages' relationship becomes an entity type as follows:

Manages (ENDstamp, Manager_Code, Sales_Area_Code, BEGINstamp)

Note that only the ENDstamp is required to provide the property of uniqueness to the entity. The author has chosen ENDstamp rather than BEGINstamp for physical implementation reasons. If the ENDstamp is given a special value and is placed first in database composite key structure, then it is a simple matter to extract all current ownership records. He also asserts that, in some circumstances, less physical IO would be needed. This is clearly an implementation rather than a modelling consideration.

In a conceptual model, if the many-to-many relationship were to be resolved by the introduction of an intersection entity, one might choose the BEGINstamp as part of the 'logical' identifier.

Where an entity has an attribute, it is really engaged in a relationship with a domain of attribute values. The bubble notation for representing attributes is, in fact, a diagrammatic shorthand. When history is introduced, it is convenient to abandon this method and explicitly represent the attribute as a relationship between an entity and a domain. In RAKE, this is done in precisely the same way as other temporal relationships, as Figure 3.7 shows:

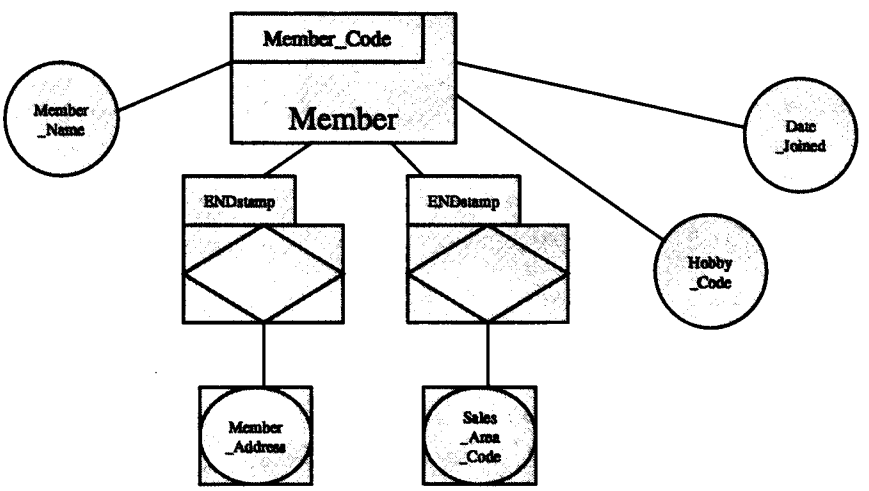


Figure 3.7 Representation of a temporal attributes in RAKE

The relationship between an entity and the attribute domain is an 'Entity-Instantiates-Attribute' relationship that RAKE makes explicit.

The relationship 'Member Has Member_Address' is entirely consistent with any other temporal relationship and could be implemented using an intersection entity as follows:

Member_Address (ENDstamp, Member_Code, Member_Address, BEGINstamp)

A complete RAKE diagram of the Wine Club is shown in Figure 3.8.

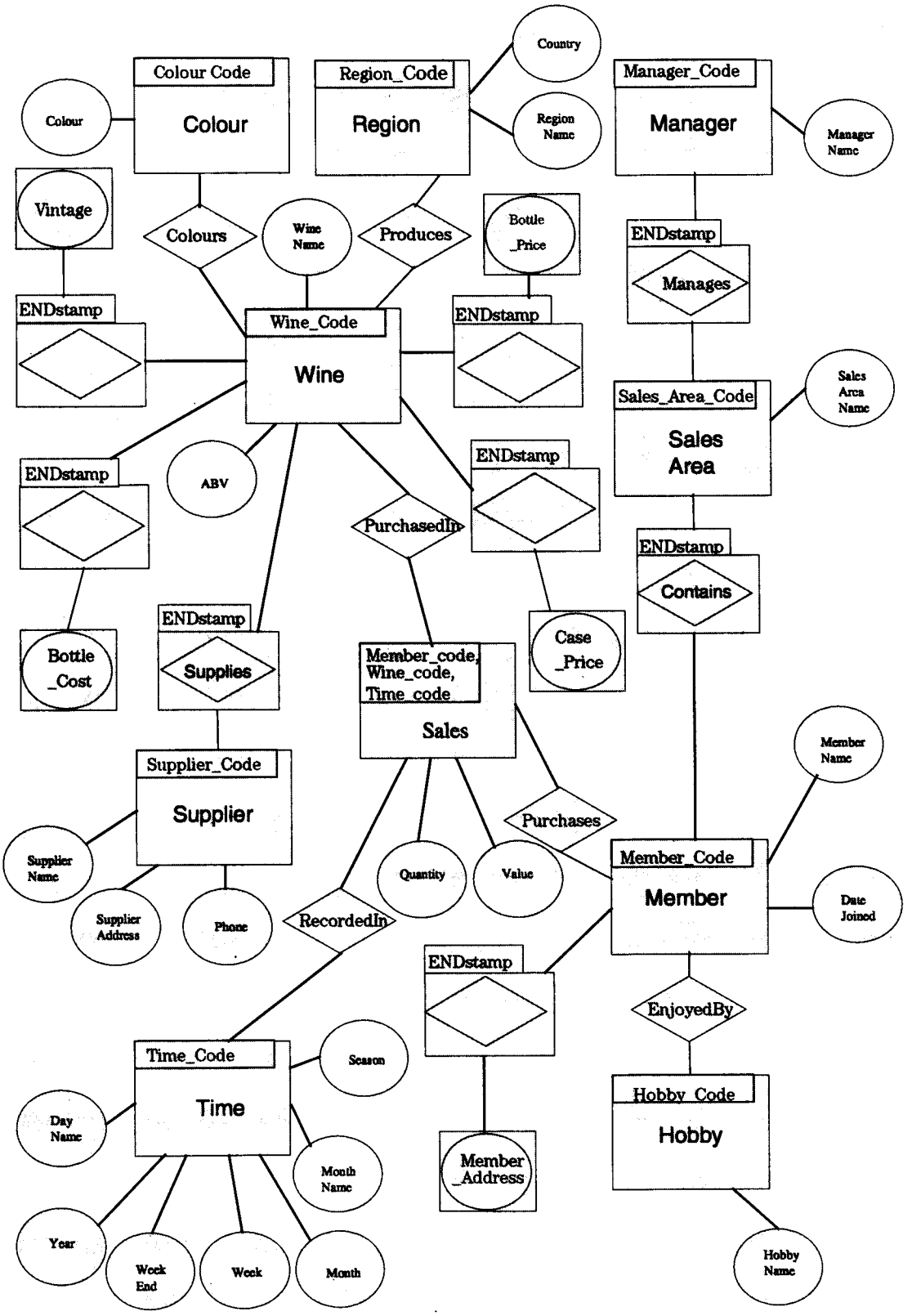


Figure 3.8 Representation of the Wine Club using RAKE

3.4 The Entity-Relation-Time Model

The Entity-Relation-Time model (ERT), developed by McBrien et al (1992), is based on the traditional Entity-Relationship model. Emphasis is placed on relationships over attributes. An Entity type is seen as entering into relationships with other Entity types and with domains. Each attribute type is treated as a set of values that engage in a relationship with the Entity type. Attributes are known as 'Value Classes' in ERT. The method separates time into a finite set of equal time instances called *ticks* where 't' is a tick. Time is bounded by restricting t to (say) a finite subset of natural numbers $0 \ldots \tau$.

A time-interval is a contiguous set of ticks. Each tick is the same length of time within a model and, therefore, each model is able to set its own granularity with respect to time.

The basic components of ERT are shown in Figure 3.9:

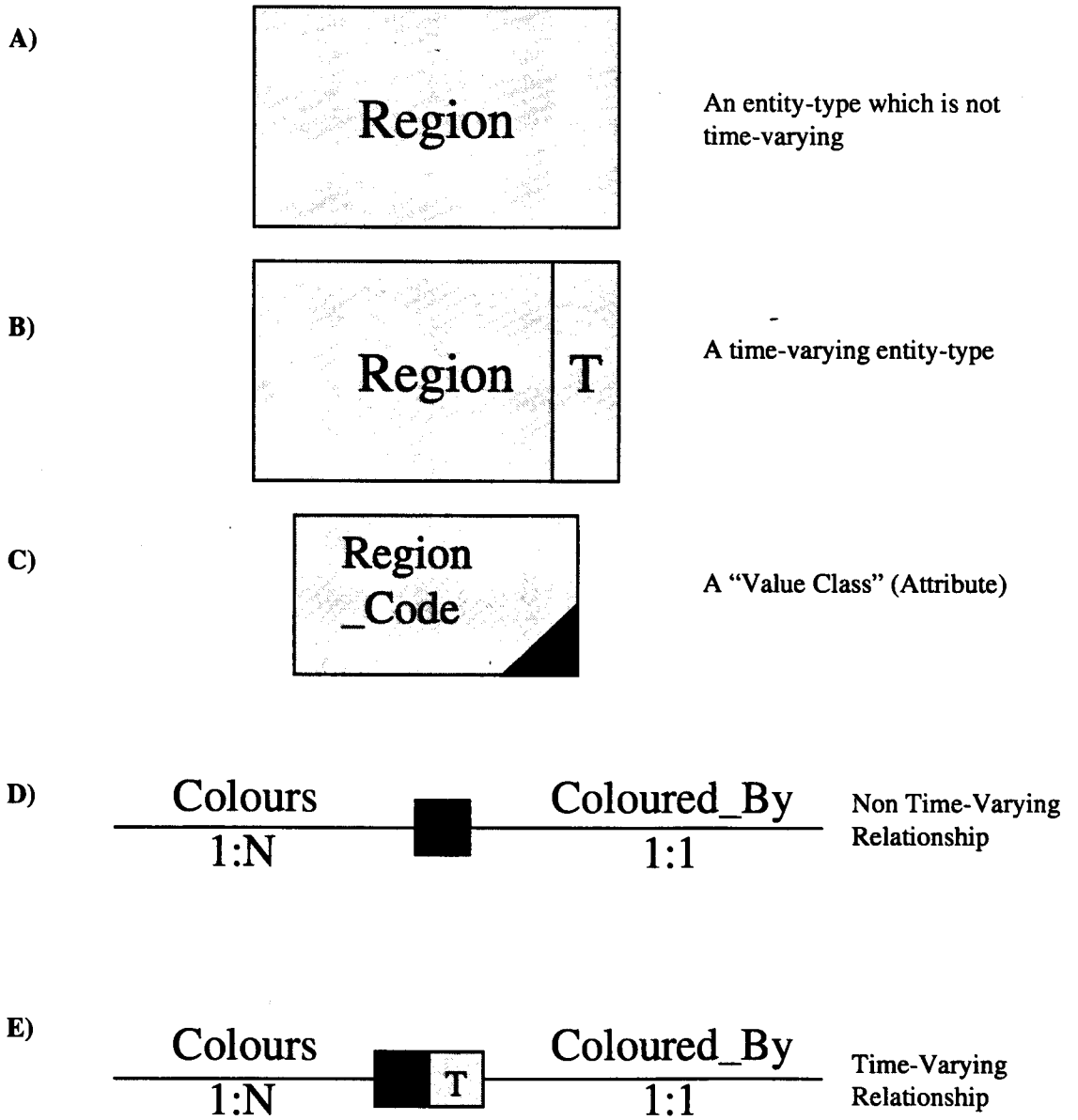


Figure 3.9 Basic ERT symbols

The standard entity type symbol (Figure 3.9 ('A')) is retained from the ER method. Entity types can be time-varying with respect to an entity's existence. If an entity type has variable existence, then the entity type symbol is marked with the 'Time' symbol 'T' (Figure 3.9 ('B')). In ERT, the addition of the time symbol to an entity type implies the addition of a time stamp. Value Classes are symbolised by a rectangle with a shaded triangle (Figure 3.9 ('C')) in the bottom right-hand corner. There are no time-variant features associated with Value Classes. Relationships are symbolised by a shaded square with a line from the entity type to the

relationship and a second line from the relationship to the other entity type or value class participating in the relationship. The two roles of the relationship are named and the degree (cardinality) and participation conditions are made axiomatic by an enumeration in the form 'min:max' as follows:

0:1 = Zero or one (optional)

0:N = Zero, one or many (optional)

1:1 = Exactly one (mandatory)

1:N = One or many (mandatory)

With this approach to modelling, a change to an attribute's value is really a change to the relationship between the entity and the set of values (domain) in the value class. The ability for the model to record changes over time is shown by the addition of a time symbol to the relationship (Figure 3.9 (E)). It is important to note that the absence of the time symbol does not imply that changes to values cannot occur, it simply means that there is no requirement to record the history. This means that old values can be overwritten by new values. As with entities in ERT, the addition of the time symbol to a relationship implies the addition of a time stamp.

In ERT the granularity of time is specific to the model. There is nowhere on the model to record the granularity in use. The actual length of time assigned to a tick would be what determines the grain. There is no means to apply different grains of time to discrete parts of the model.

As has been shown, the addition of a time symbol to either an Entity type or a Relationship type signifies the need for temporal support. However, there does not appear to be any method for separating items that will never change from items that may change but for which there is no requirement to record the history.

The complete ERT diagram for the Wine Club is shown in Figure 3.10:

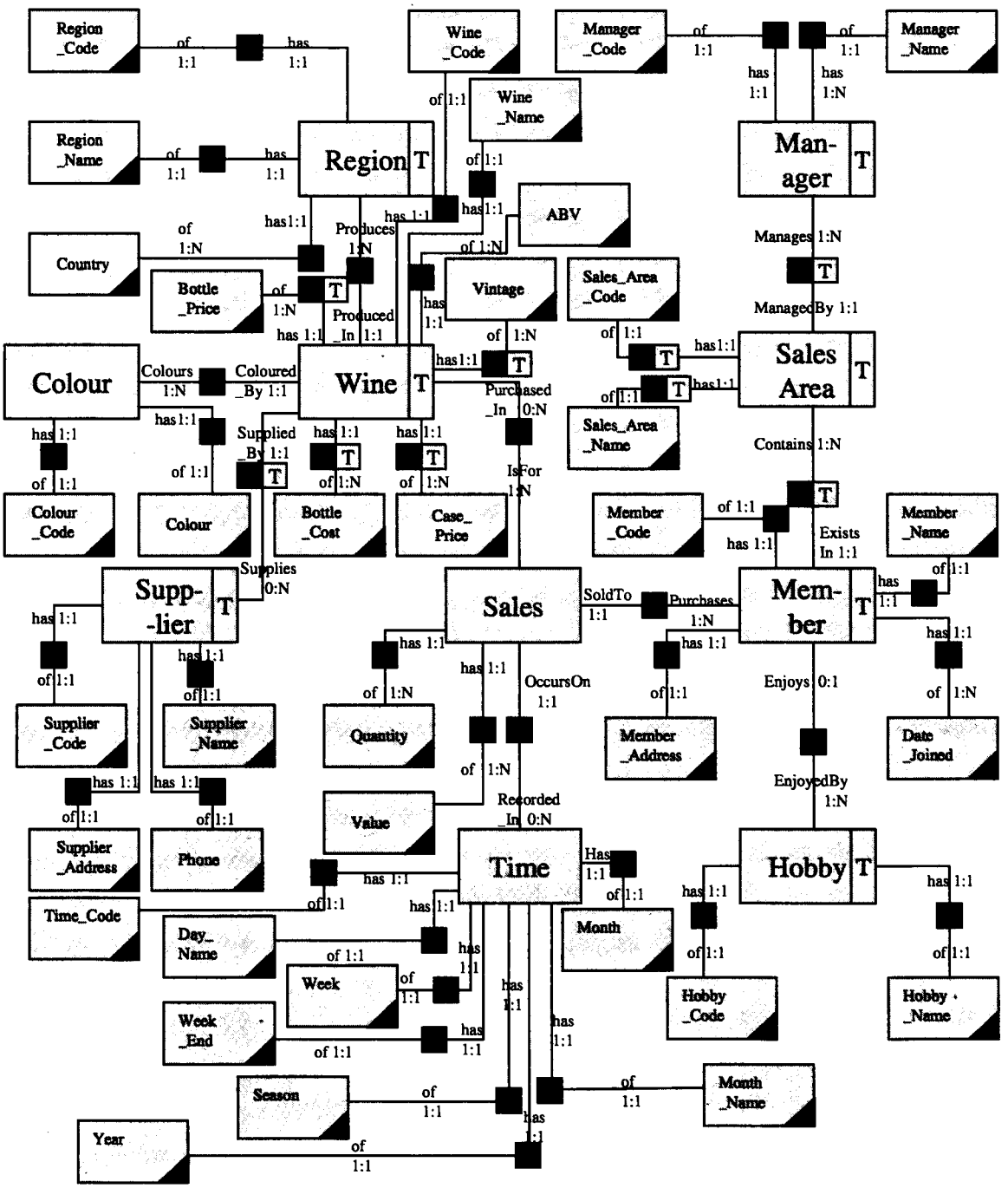
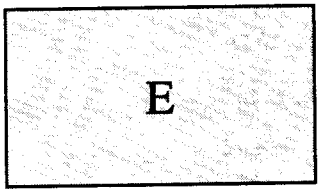


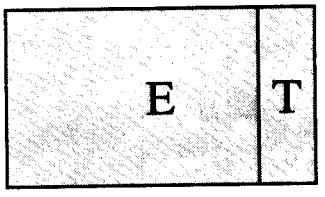
Figure 3.10 Representation of the Wine Club using ERT

The ERT method also provides some mapping rules from the diagrammatic components to a Relational Schema, as shown in Figure 3.11:

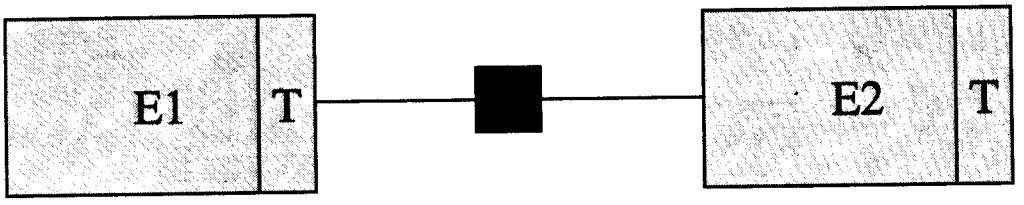
ERT Mapping to Relational Schema



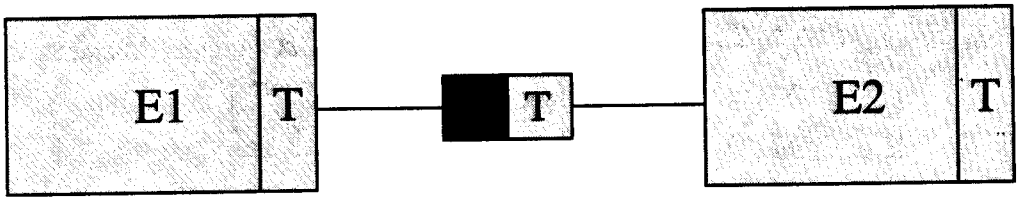
E(E#, att-1, att2, . . .)



E(E#, Start, End, att-1, att2, . . .)



R(E1#,E2#)



R(E1#, E2#, Start, End)

Figure 3.11 ERT mapping to relational logical model

The mapping rules show that where an entity or a relationship is temporal, a start time and an end time are implicit in the logical model.

3.5 The Temporal Enhanced Entity Relationship Model

The Temporal Enhanced Entity Relationship (TEER) model is an adaptation, by Elmasri and Wuu (1990), of the EER model, described by Elmasri and Navathe (1994), to include the temporal dimension. The graphical notation of the EER model is retained but the meaning of the components is altered so that all components are temporal. The components of the method are entirely consistent with the EER method and there is nothing, in diagrammatic terms, to differentiate TEER from EER.

Each component is associated with a time interval $[t_1,t_2]$ that is defined to be a set of consecutive equidistant instants of time. The interval starts at t_1 and continues until t_2. The database contains historical information within the time interval [0,*now*] where 0 represents the beginning of time, insofar as the database is concerned, and *now* represents the ever expanding current time. The time interval $[t_1,t_2]$ is constrained to being a subset of the interval [0,*now*].

The support for time is implicit in the model. Each entity (e) within an Entity Type (E) is associated with a temporal element $T(e) \subseteq [0,now]$. The temporal element gives the lifespan of the entity and can be a continuous time interval or a union of a number of disjoint time intervals to allow for the representation of discontinuous lifespans. The implementation of an entity lifespan is achieved through the creation of a system generated, generalised *SURROGATE* key attribute that is unique within the entire database.

Attributes, including the *SURROGATE* attribute, are treated as follows: The temporal value of each attribute of the entity $A_i(e)$ is a mapping from a temporal element to a value from the domain of the attribute ($A_i(e)$: $T(e) \rightarrow dom(A_i)$). There are constraints that force temporal assignments of non-*SURROGATE* attributes to fall within the temporal assignment of the *SURROGATE* attribute to ensure integrity within the database.

An example of a Wine entity, using TEER, is shown in Figure 3.12:

Surrogate = {[1 Jan 1997,now] → surrogate_id}

Wine_Name = {[1 Jan 1997,now] → Beaujolais Villages}

Vintage = {[1 Jan 1997, 21 Jan 1998] → 1996,

[22 Jan 1998,now] → 1997}

ABV = {[1 Jan 1997, 21 Jan 1998] → 12,

[22 Jan 1998,now] → 13}

Bottle_Price = {[1 Jan 1997, 21 Jan 1998] → 3.49,

[22 Jan 1998,now] → 3.99}

Case_Price = {[1 Jan 1997, 21 Jan 1998] → 37.69,

[22 Jan 1998,now] → 43.09}

Bottle_Cost = {[1 Jan 1997, 21 Jan 1998] → 2.32,

[22 Jan 1998,now] → 2.66}

Figure 3.12 Example Wine entity using TEER

Relationships are treated in much the same way. Each relationship instance r is associated with a temporal element T(r) that gives the lifespan of the relationship instance. The temporal value of each attribute of the relationship $A_i(r)$ is a mapping from a temporal element to a value from the domain of the attribute ($A_i(r)$: $T(r) \rightarrow dom(A_i)$).

The granularity of time (described as 'the distance between two consecutive time instances') in TEER is determined by the application. There is only one level of grain per database.

In TEER, all components (Entities, Attributes and Relationships) are regarded as temporal.

As the TEER model is based on the Enhanced Entity-Relationship method, it does not add any new symbols to deal with time. Instead it redefines all the components to make them temporal in nature. So a TEER diagram will look precisely the same as an equivalent EER diagram. It is the underlying meanings that have changed

The TEER diagram for the Wine Club is as follows:

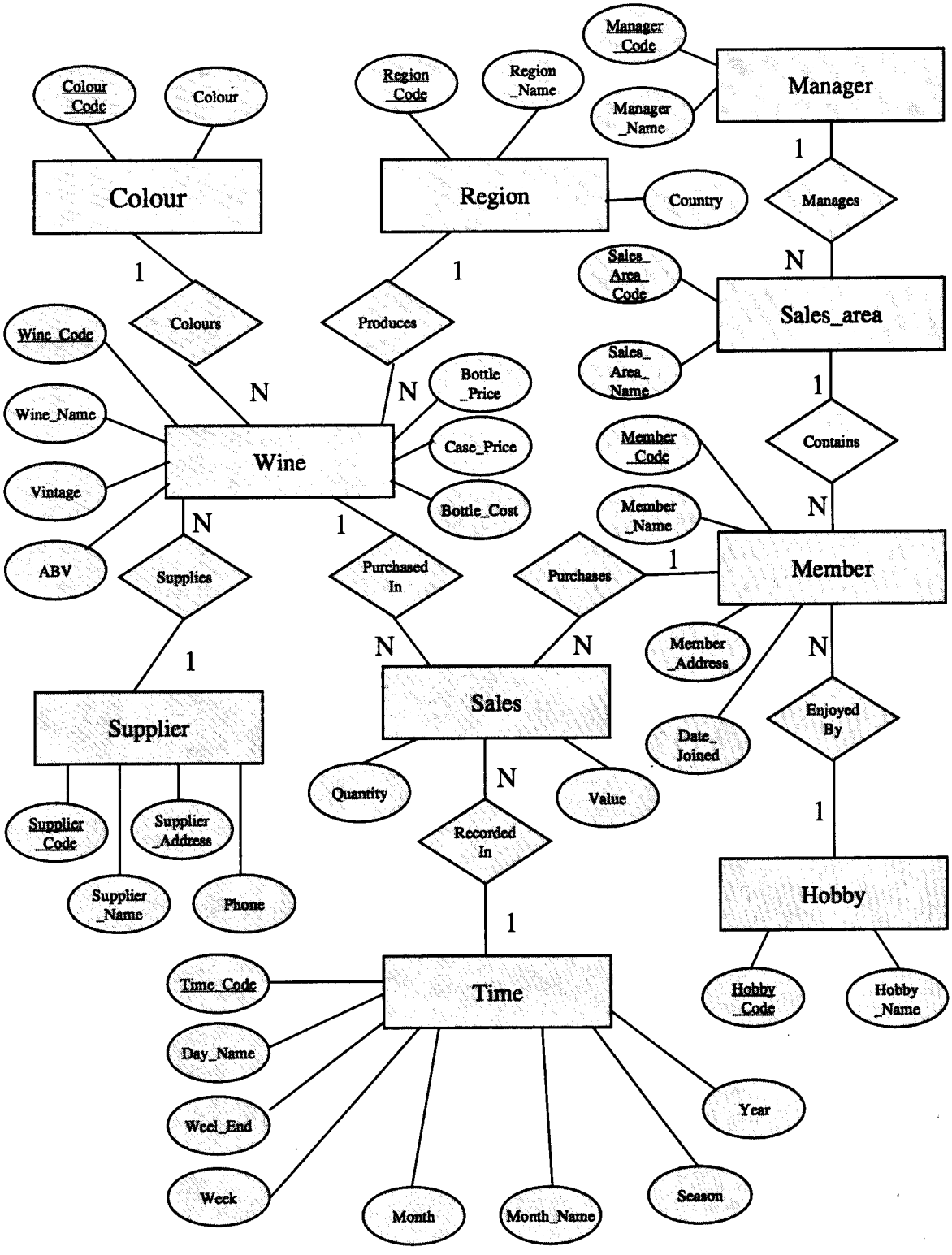


Figure 3.13 Representation of the Wine Club using TEER

3.6 The Semantic Temporal Enhanced Entity-Relationship Model

The Semantic Temporal Enhanced Entity-Relationship Model (STEER), developed by Elmasri and Kouramajian (1993) and Elmasri et al (1990), is a further enhancement of the EER model. In particular it distinguishes between, what the author calls, conceptual and temporal entity types and relationship types. This distinction, it is asserted, is missing from other methods and represents a fundamental shortcoming in those methods.

The method describes a clear distinction between conceptual entities and temporal entities. Conceptual entities are entities that are of interest to an organisation even if the conceptual components of the entity play no part in the operation of the organisation. The example given in the paper is of employees. An employee exists in the real world as a person but only becomes of interest to a company when it hires them as an employee. After the employee has left the company, they may remain an object of interest as long as the company still wishes.

The main characteristic of a conceptual entity is that it has an *'existence time'*. The existence time of a conceptual entity starts when the entity is first materialised and this is called the *start time* ST. There is no end point to the existence time of a conceptual entity. The end time is given as infinity, so the existence time is given as ET = [ST, ∞]. So, in STEER, conceptual entities are held forever, once registered. An example of a conceptual entity in the Wine Club model is the Wine entity type. As the model asserts, wines exist outside of our application, and details of wines that are not stocked by the club may still be of interest to the club. In a sense, this is an extension to the kind of information that the Wine Club routinely holds about wines.

Temporal entities, in STEER, are defined as *roles*. For instance: in the Wine Club, a wine may play a role in the organisation once it becomes part of the product range and is, in effect, a *'stocked wine'*. Roles have a lifespan that must be a subset of the existence time of the conceptual entity. There are several axiomatic constraints pertaining to roles and conceptual entities.

The attributes of a conceptual entity are independent of its attributes as a role. The attributes of a conceptual entity are inherited by the role and the role may have attributes of its own. For instance, the conceptual attributes of a wine would be its name, region, colour etc. while the

temporal attributes of the role of a stocked wine would be the cost price, selling price, vintage etc.

Conceptual entities are restricted to non-temporal attributes. Roles are not allowed to own non-temporal attributes (except by inheritance). The value of a non-temporal attribute of an entity holds over the entire existence time of the entity.

As with some other methods, each entity is associated with a system allocated *SURROGATE* identifying attribute that is non-temporal. Temporal attributes are treated in precisely the same way as the TEER model by Elmasri and Wuu (1990).

Relationships are treated in the same way as attributes. Non-temporal relationships have an existence time ET = [ST, ∞] and temporal relationships have a lifespan.

In the graphical model, the following rules apply:

1. Conceptual entities are transparent rectangles
2. Entity roles are filled rectangles
3. Non-temporal relationships are transparent diamonds.
4. Temporal relationships are filled diamonds
5. Non-temporal attributes are enclosed in circles
6. Temporal attributes are enclosed in ellipses

Within STEER, the granularity of time is application dependent. There is one level of grain per application.

The STEER diagram for the Wine Club is shown in Figure 3.14:

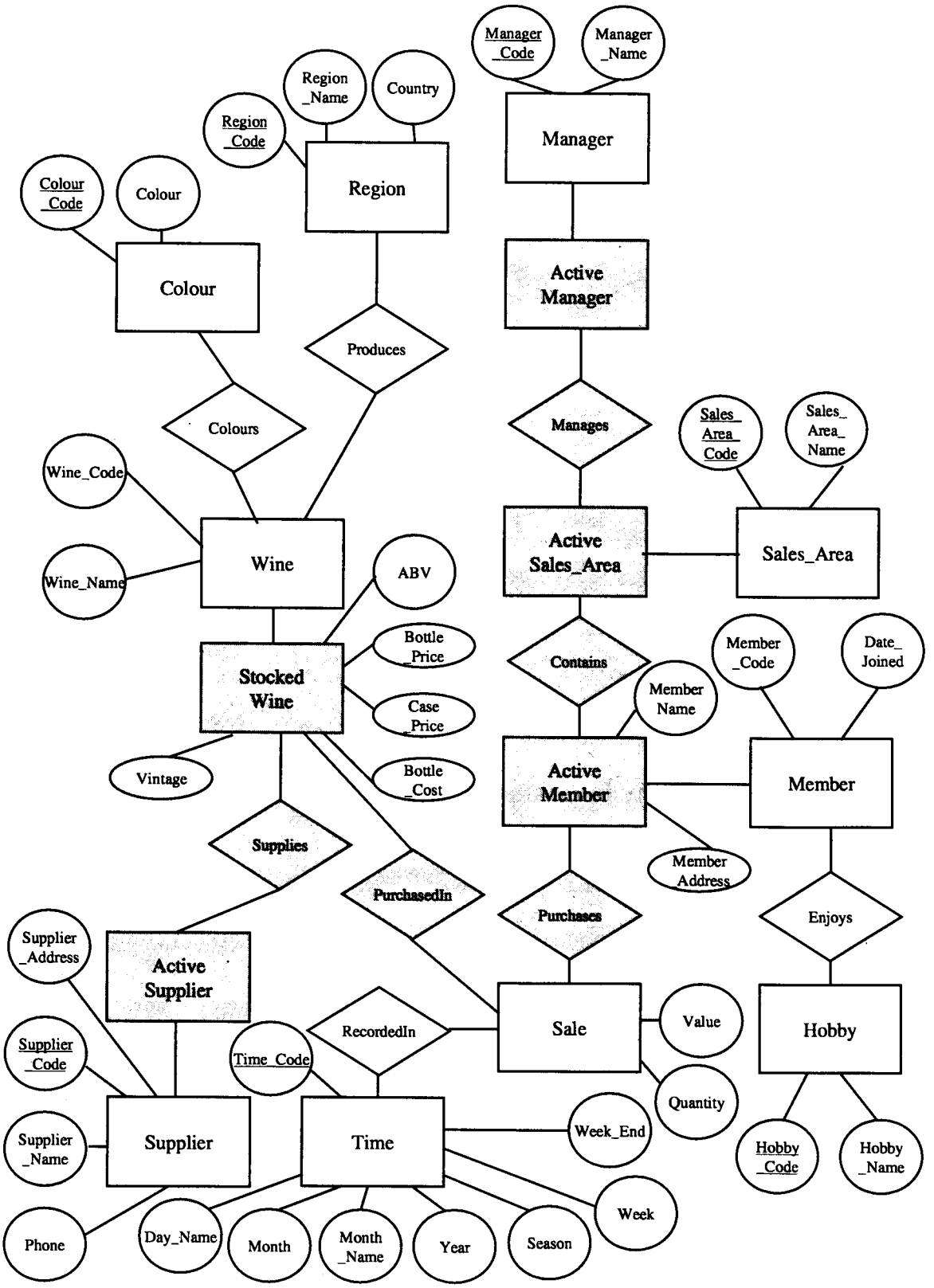


Figure 3.14 Representation of the Wine Club using STEER

The STEER method recognises that some attributes, related to an entity, are static over time while others are inclined to change over time. STEER deals with this by separating the fixed attributes from the variable attributes and placing them into separate entities. For instance, in the example given, a 'person', which is a conceptual entity has attributes of Name and Social Security Number (the assumption made is that the name never changes). A temporal role called 'living person' has attributes of address and telephone number. This is saying that a person, while they are alive, may have many addresses and telephone numbers. This could be modelled, in ER modelling, by the use of a one-to-many (1:n) relationship between person and living person. However, as has been said previously, the ER method does not explicitly support time so this method does solve that particular problem. The conceptual entity 'person', according to the syntax rules, would have a start date but no end date. The temporal role called 'living person' would have a start date and an end date. It is stressed that these existence times do not necessarily correspond directly to dates of births and deaths but more to when they become of interest to the organisation. The practical purpose of the single start date on a conceptual entity is not clear. The diagram makes it clear which entities are conceptual and which are temporal, so the purpose of having an open ended period cannot be to identify these different classes of entity.

3.7 The Temporal Entity Relationship Model

The Temporal Entity Relationship Model (TER), by Tauzovich (1991), is based on the Entity Relationship model. It introduces a new way of expressing historical relationships. It does not extend to the treatment of time varying attributes.

TER extends the cardinality (degree) of relationships by defining two new cardinalities:

1. Snapshot Cardinality. This is the number of instances that an entity can participate in its relationship with another entity at any single point in time.

2. Lifetime Cardinality. This is the number of instances that an entity can participate in its relationship with another entity over the lifetime of the entity.

For instance, in the Wine Club, the relationship between the entity types 'Wine' and 'Supplier', the Snapshot cardinality would show a many-to-one relationship from Wine to Supplier. This

indicates that, at any single point in time, a Wine can be supplied by exactly one Supplier and that a Supplier can supply zero, one or more Wines. Over time, however, the cardinality at the 'one' end of the relationship may change to 'many' as the Wine Club switches Suppliers during the ordinary course of trading.

In TER, cardinalities are expressed in two ways, one for the snapshot and one for the lifetime. These are written as S[<minS>, <maxS>] and L[<minL>, maxL>] respectively, where minS is the minimum snapshot cardinality, maxS is the maximum snapshot cardinality, minL is the minimum lifetime cardinality and maxL is the maximum lifetime cardinality.

The method contains some axiomatic rules:

maxS > 0

maxL > 0

minS <= maxS <= maxL

minS <= minL <= maxL

The traditional cardinalities are redefined as follows:

One-to-one is a relationship in which maxL = 1 in both directions.

One-to-many is a relationship in which maxL = 1 in one direction and maxS > 1 in the other.

Many-to-many is a relationship in which maxS > 1 in both directions.

TER introduces a new cardinality: *OneT* – meaning *one at a time.* This introduces three new relationship classes:

One-to-oneT is a relationship in which the cardinality is *one* in one direction and *oneT* in the other.

OneT-to-oneT is a relationship in which the cardinality *oneT* in both directions.

Many-to-oneT is a relationship in which the cardinality is many in one direction and *oneT* in the other.

There is no need to introduce a *manyT (many at a time)* classification as *manyT* has the same meaning as *many*.

The treatment of historical information depends upon the frequency of access of the history. In TER, there are three classifications, of relationship histories, based upon frequency of access:

1. Never. Either the data never changes, or its values are overwritten

2. Occasional. It is stated that: storing historic data together with frequently accessed current data is inefficient. It is also stated that the design process is simplified if the current and historic data are separated at the conceptual level. This is termed the *separate history* approach.

3. Frequent. Where the frequency of access to historic data is high (an example is given as a hospital patient's temperature chart), then a *merged history* approach is recommended that involves storing current and historical data together.

Based on the nature of the example, the treatment of historical information in a data warehouse would be classified as frequent because most queries would have to take account of historical values. However, the historical relationships in a dimensional model have more to do with consistency of information than with frequency of access. Although there may not be a high volume of accesses, almost all of them will require access to both current and historical information. The TER representation of the Wine Club is shown in three diagrams:

1. Unresolved

2. Resolved using Separate History

3. Resolved using Merged History.

There is no reference to granularity of time in TER.

There are three levels of diagram in the TER method. These are shown as follows:

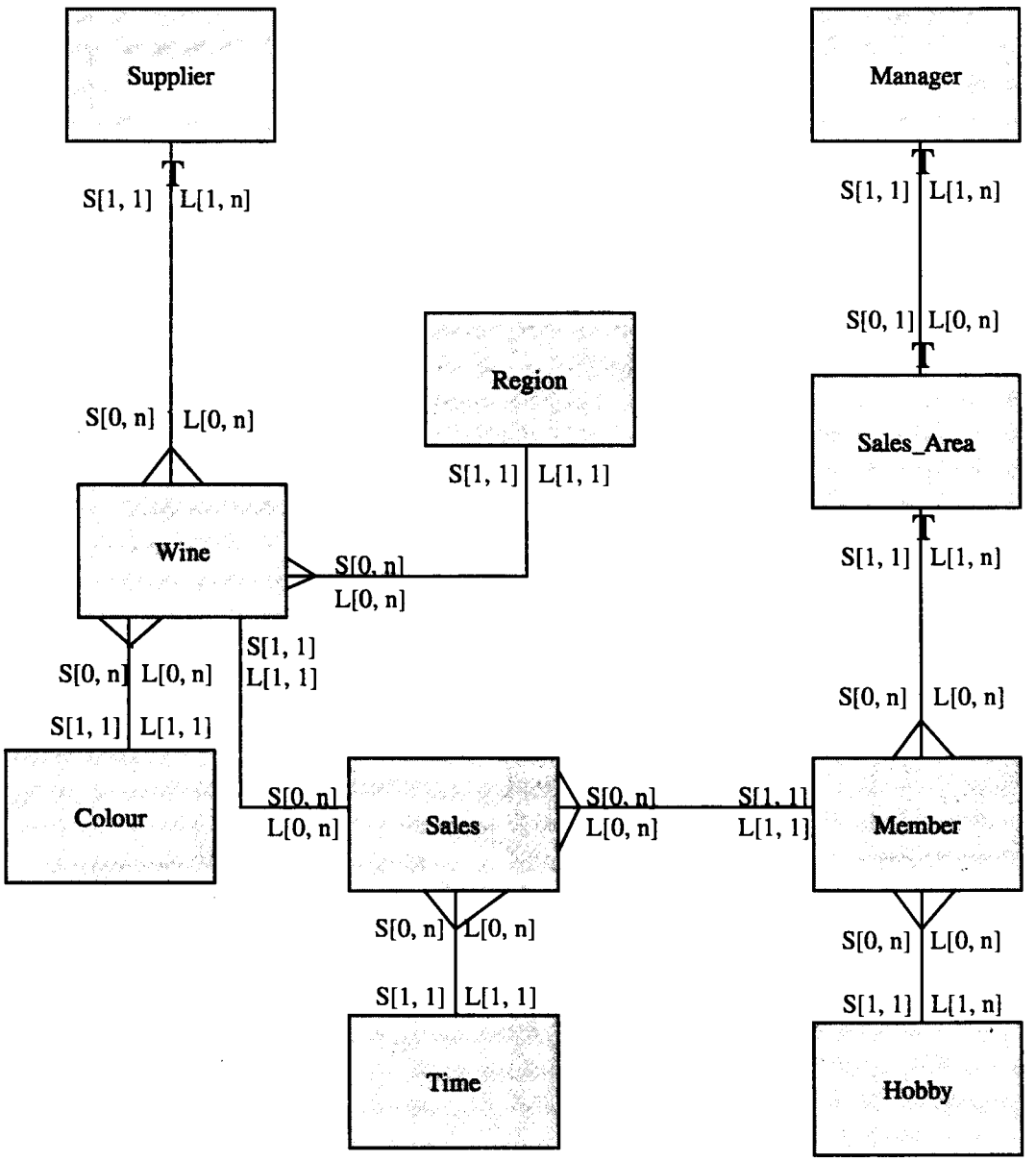


Figure 3.15 Representation of the Wine Club using TER (unresolved)

Figure 3.15 is the basic TER model before any decision is made whether to resolve it via the separate history or merged history approach. In the case where no history is required, it is recommended, by the author, that the model is converted to a traditional ER diagram by ignoring the lifetime cardinalities.

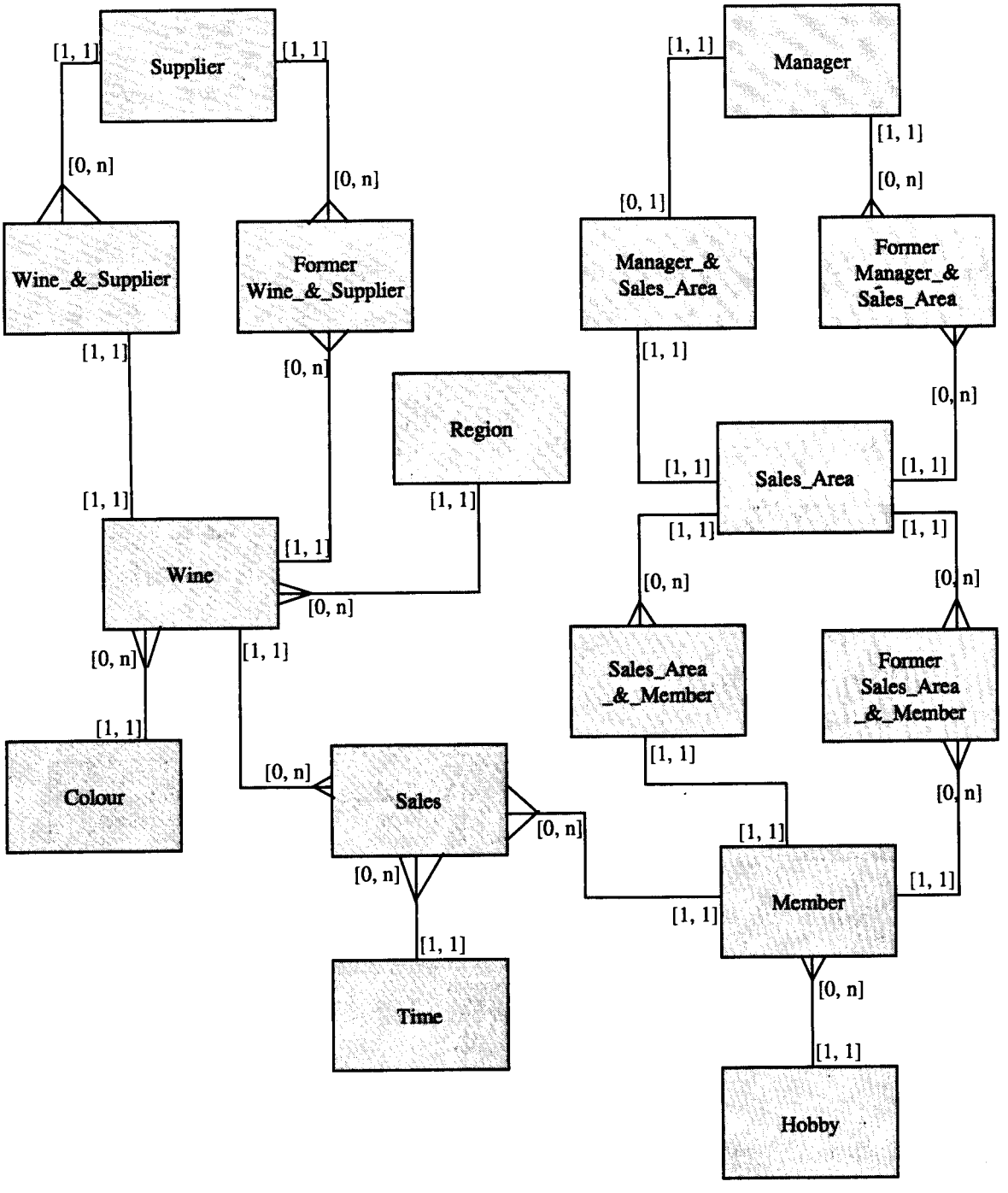


Figure 3.16 Representation of the Wine Club (resolved using separate history)

Figure 3.16 has been produced from the original unresolved diagram using a set of transformation rules supplied by the author. It involves the introduction of intersection entities to model the cardinalities that have been changed from one-to-many to many-to-many due to the introduction of history. The separate history approach exists for performance

reasons. The author asserts that storing current data separately from historical data is more efficient. In this method, two intersection entities are created for every relationship that requires history to be recorded.

The merged history approach combines the two intersection entities into a single entity. This means that the current and historical data are stored together. There exists an equivalent set of transformation rules to enable the merged history diagram to be developed from the original unresolved diagram. This is shown in Figure 3.17

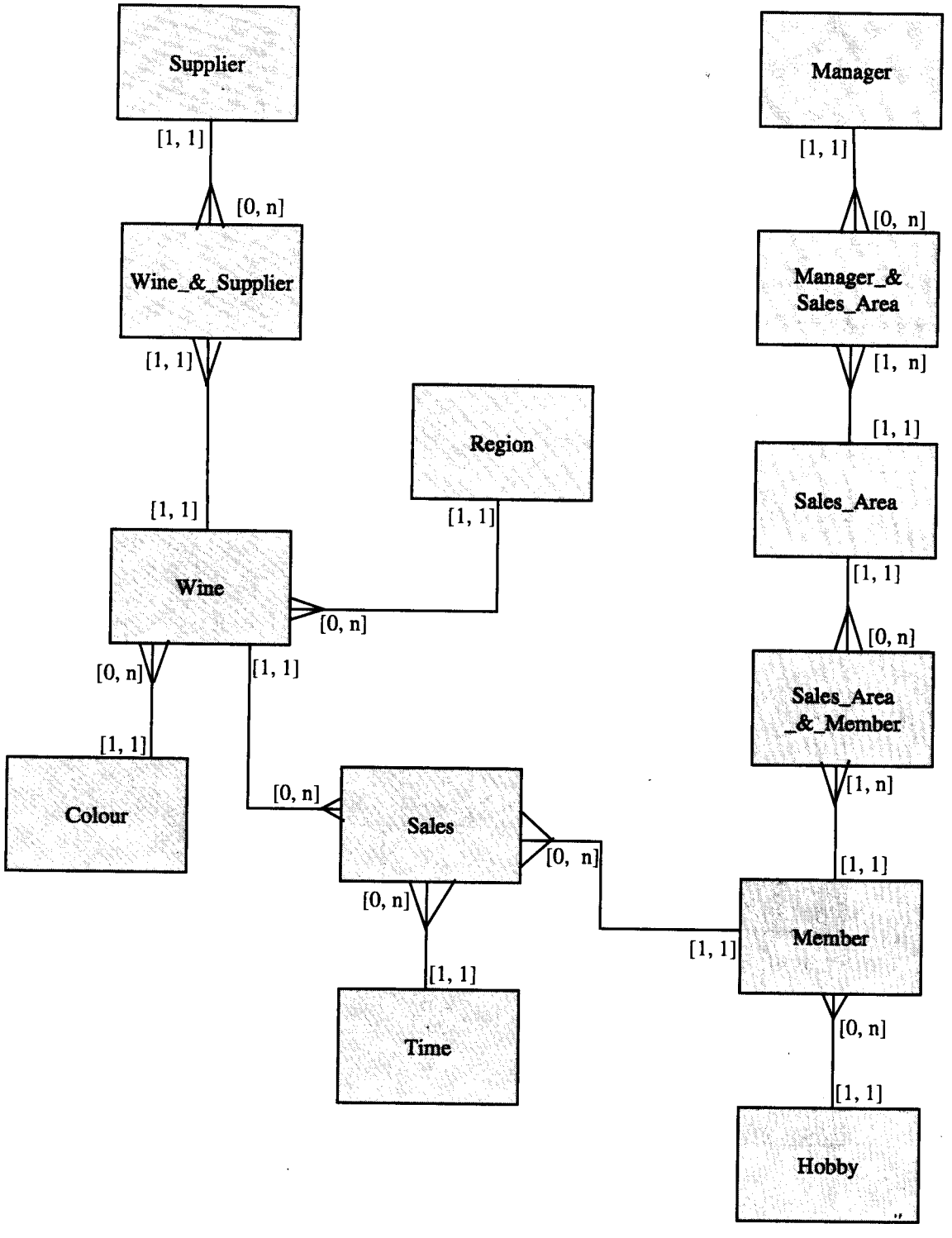


Figure 3.17 Representation of the Wine Club (resolved using merged history)

3.8 The Temporal Enhanced Entity-Relationship Model

The Temporal Enhanced Entity Relationship Model, presented by Lai et al (1994), is called TEER^2 (the superscript is added because the authors originally named their method as TEER which is the same as a previous model) and extends the EER model by adding transaction time and valid time attributes to tuples. It assumes that, at the implementation level, the target database technology will be relational.

The attributes of an entity type are extended to include time stamps. The method seeks to resolve confusion by defining four approaches to include time stamps in a model:

1. Time-Point Tuple Time Stamping. A time stamp is held at 'tuple' level.

2. Time-Interval Tuple Time Stamping. Two time stamps, marking the start and the end times, are recorded at the 'tuple' level.

3. Time-Point Attribute Time Stamping. A time stamp is held for each attribute.

4. Time-Interval Attribute Time Stamping. Two time stamps, marking the start and the end times, are recorded for each attribute.

In TEER^2 the adopted time stamping approach is Time Interval Tuple Time Stamping. The reason was 'to preserve the simplicity and ease of implementation of the model'. TEER^2 introduces time elements that are a subset of (0, UNTIL). In this method, 0 is considered to be the start point of time and UNTIL is an expanding time variable greater than the system time. It can be thought of as an extension to the NOW variable. This enables the future to be modelled beyond the NOW attribute. Even though data warehouses are concerned with history, there sometimes exists the need to allow future data to be entered for budgets and forecasts or, as Kimball (1998e) observes, where there are payments in advance such as subscriptions.

In the TEER^2 model, each entity has a life span. The life span is represented by a time interval that consists of a start time and an end time. Any change to the value of any of the attributes belonging to the entity result in the creation of a new entity that contains the existing values of the unchanged attributes and the new values of the modified attributes. The old entity is

updated with the end time being set to T-1 where T is the same as the NOW attribute. The value of T is determined by the granularity chosen for the model.

The table in Figure 3.18 describes the life span of a wine from the Wine Club database.

Valid Time	Wine Code	Wine Name	Year	A B V	Bottle Price	Case Price	Bottle Cost	Trans Time
01/01/1997, 12/09/1997	2101	Claret Lot 278	1995	12	3.49	39.79	2.18	24/12/1996
13/09/1997, 28/10/1998	2101	Claret Lot 278	1995	12	3.59	40.92	2.25	13/09/1997
29/10/1998, 14/03/1999	2101	Claret Lot 278	1995	12	3.59	40.92	2.10	29/10/1998
15/03/1999, UNTIL	2101	Claret Lot 278	1995	12	3.79	43.20	2.25	15/03/1999

Figure 3.18 Fragment of the life span of a wine

The record shows the life span of wine code 2101. It was first brought into the club on 1st January 1997 and this is shown by the valid time (start). The valid time (end) would have been given a value of 'UNTIL'. The transaction time shows that it was introduced into the database on 24th December 1996.

There was a price increase that took effect on the 13th September 1997. The first tuple is 'closed' by updating the valid time (end) to 12th September 1997 which is the last day that the old details were true. A new tuple is created that contains the old, unchanged attribute values together with the new values.

On 29th October 1998, the supplier of the wine was changed to enable the club to take advantage of the cheaper purchase price. So the bottle cost is updated in the next tuple.

Finally, on the 15th March 1999, another price increase took place. The valid time (end) is set to 'UNTIL' which is how the system would be able identify the current tuple.

The relationships are treated in much the same way. The table in Figure 3.19 is an example of the 'Supplies' relationship that follows from the example in Figure 3.18.

Valid Time	Supplier Code	Wine Code	Trans Time
01/01/1997, 28/10/1998	2122	2101	24/12/1996
29/10/1998, UNTIL	3442	2101	29/10/1998

Figure 3.19 Fragment of the lifespan of a relationship

This relationship table shows that the wine was supplied by supplier 2122 from the start of the life span of the wine until 28th October 1998 after which a new supplier was chosen (supplier 3442) and the new supplier remains as the current supplier.

In the TEER^2 model, attributes are not modelled distinctly with respect to time.

The granularity of time is set on a 'per application' basis. There is no support for multiple grains of time within an application,

The TEER^2 model does not add any new features to the EER model insofar as diagramming is concerned. So a diagram for the Wine Club using the TEER^2 model has not been produced. The diagram produced for the first TEER method (Figure 3.13) would be the same as that for TEER^2.

3.9 TERC+: A Temporal Conceptual Model

The TERC+ method is presented by Zimanyi et al (1997) and has the following objectives:

1. Completeness. The ability to describe all usual temporal applications. The capability to combine temporal and non-temporal elements. Support for instantaneous events as well as other elements.

2. Soundness. Formal definitions avoiding ambiguities.

3. User orientation. Easily understood by users.

4. Orthogonality. Constructs must be as independent as possible from one another to make the modelling language easy to understand and use.

5. Implementability. The modelling language must translate directly into logical data models of existing DBMSs.

6. Full Operationality. It must include a data manipulation language.

In principle, TERC+ is based on the ER method. It has some additional features to provide temporal support for entities, attributes and relationships.

The method describes two ways of adding time to information.

Timestamping of entities, attributes and/or relationships. This enables entities and relationships to have an existence lifespan and attributes to have values during periods of time.

Relative positioning of activities, or events, in time. This allows the determination of a historical context. An example might be the purchasing behaviour of a Wine Club member or a class of members.

Defining an entity type as temporal instructs the DBMS to record the existence of an entity for special temporal treatment. So, although claiming to be a conceptual model, TERC+ implies some implementation considerations. TERC+ identifies three states that an entity may be in at any one time:

1. Active

2. Suspended

3. Dead

Entities are declared as temporal by the placing of a time symbol ◷ into the entity-type rectangle. Similarly, the same symbol is used to declare relationships and attributes as temporal. Recognition is given to certain classes of temporal relationships, as will be shown later.

Non-temporal attributes may be constant over time, or may change without the application being interested in recording the history. Therefore there is no means of differentiating between those two.

Cardinality (degree) of relationships is determined as follows:

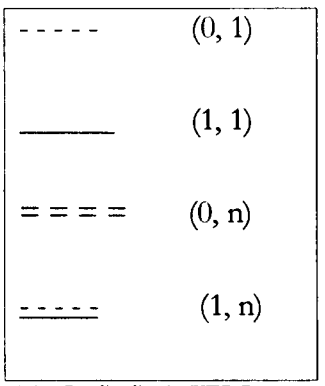


Figure 3.20 Cardinality in TERC+

TERC+ claims to support multiple time granularities as follows: 'When defining an attribute as temporal, its granularity must be specified'. The method does not explain exactly how this is represented within the model.

A full TERC+ diagram for the Wine Club is shown in Figure 3.21:

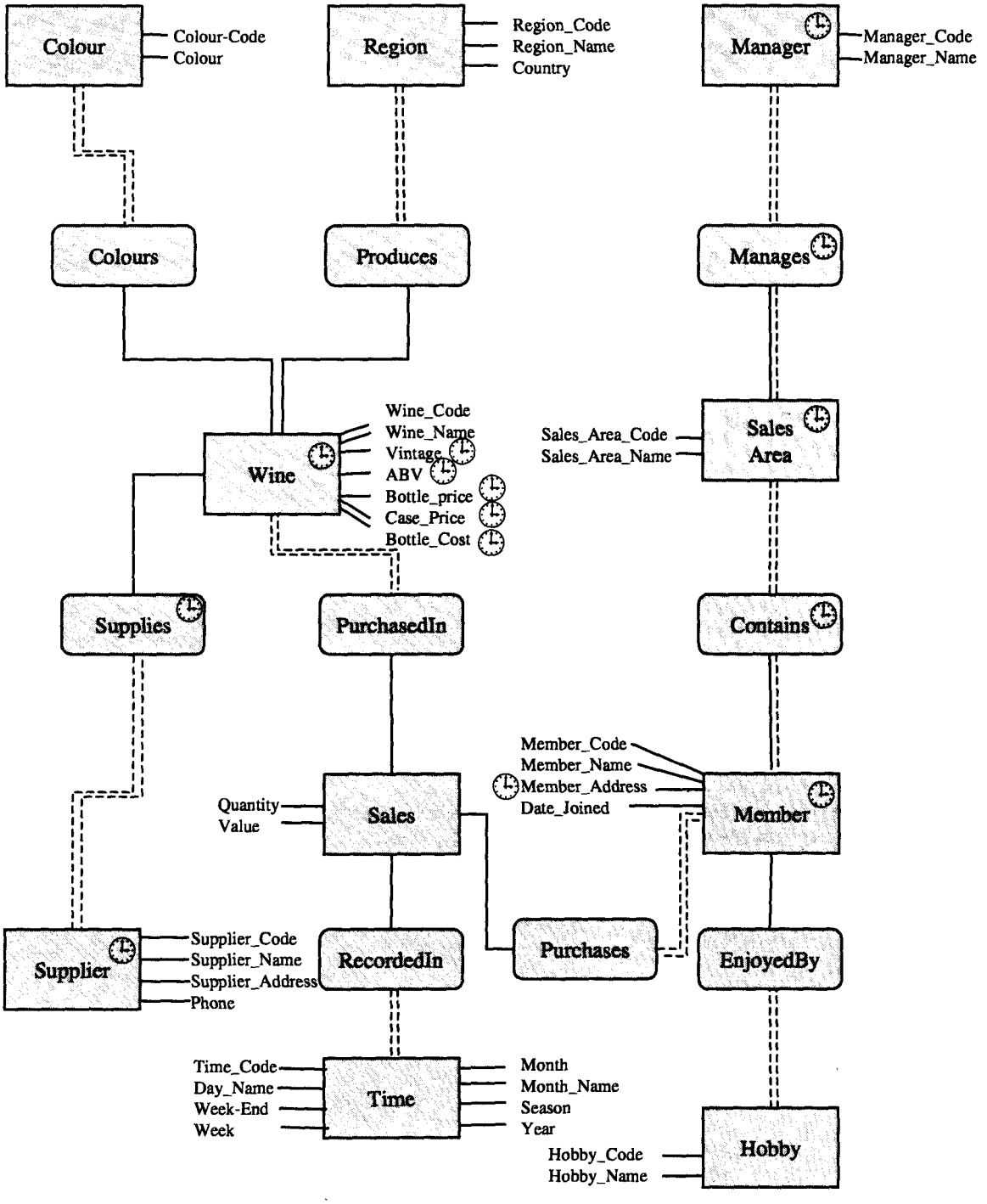


Figure 3.21 Representation of the Wine Club using TERC+

As can be deduced from the diagram, the main components are very similar to traditional ER models including entities, relationships and attributes.

There are several special classes of relationship, one of which is called 'timing' relationships. The purpose of timing relationships is to show, on the diagram, the particular timing properties associated with the relationship.

The diagram below shows a relationship example from the Wine Club that possesses the 'meets' property.

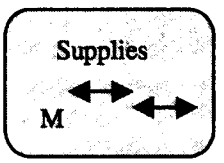


Figure 3.22 Specifying time functions in TERC+

This diagram shows the Supplies relationship for the Wine Club. It is a temporal relationship with a timing classification of 'Meets'. Within the symbol, the 'M' means 'Meets' and the arrows show that the timing of relationship changes are that they should meet. This means that there are to be no gaps and no overlaps in the supplies relationship.

This example is selected for illustrative purposes only. In the Wine Club it is possible to have gaps in the supply of wine while the club changes one supplier for another.

Other timing classifications are: (before, equals, overlaps, during, starts, finishes). The main requirement in data warehousing is for ***no overlaps*** in order to prevent double counting. Although this option is missing from the model, it could easily be added.

3.10 Summary of review of conceptual modelling methods

In this summary, the comments are intended to relate to the methods as being suitable for use in *dimensional modelling* only. There is no attempt to assess the methods for general use.

Almost all of the methods reviewed are based, with varying notations, on the traditional entity-relationship method. The methods all augment the traditional approach to include temporal aspects. This is achieved either by adding new symbols and features or by altering the meaning of existing entity-relationship symbols or features.

There is a great deal of commonality in the approach to implementing support for time. The consensus supports the idea of timestamping. There are differing views as to whether lifespans should be held, in the form of a start and end timestamp, or whether a single timestamp is sufficient. Similarly, there are differing views about whether the temporal support should be applied to the entity level or the attribute level. Also, the different methods have differing views about the need for valid time and transaction time support.

Before beginning an examination of conceptual modelling methods, it is important to understand the objectives of conceptual modelling in data warehouse projects. There are three main objectives that a conceptual model should satisfy:

1. It should be possible to express the temporal requirements for the information. Time should be regarded as an inherent part of data warehouses. Each data warehouse object (fact, dimension, attribute and relationship) has an associated temporal property that should be accounted for.

2. The model should enable the organisation's information requirements to be expressed and, as data warehouses are usually developed on behalf of non-technical people, it should be easy to comprehend. Therefore the diagram must not be overly cluttered with symbols and notation must not comprise too many different types of symbol.

3. As the most common approach to the representation of information in data warehouses is to use a dimensional schema, the model should enable dimensional schemes to be

expressed. Further, the essential radial nature of the dimensional model should remain intact irrespective of the amount of information displayed on the diagram.

All of the temporal models under review recognise the need to provide temporal support for data that changes over time and that the historical values must be preserved. However, only four out of the seven temporal models identified a second type of data. RAKE and ERT assume that a second type of data exists for which there is no requirement to record any changes over time. This means that changes are implemented by destructive update and that the old, now changed, data is lost and adopts the appearance that it never existed while the new data has the appearance of having always existed. An example of this in the Wine Club database is the Member's name attribute. If a change to a member's name is implemented, it is appropriate to overwrite the previous name as the club does not need to recall previous names of members. STEER assumes that a second type of data exists but in this case the values will never change and are, therefore, assumed to be permanent. An example of this in the Wine Club is the 'Region' in which wines are grown. Once a region has been established it is unlikely to change. TERC+ also recognises a second case but does not distinguish between the two types. In TEER and $TEER_2$, the assumption is made that all data is temporal and requires history to be faithfully recorded.

When temporal database management systems are generally available the distinctions will become less important because it is assumed that all database objects will receive complete support for time. However, at the present time such a facility does not exist and temporal support has to be provided explicitly on a case by case basis. The implementation of temporal support is expensive in computer resource terms and should be provided only where it is needed. In data warehousing there is a further reason for the distinction that relates to the design of the data warehouse population processes that extract changed data from the source systems of the organisation. Where attribute values are not expected to change over time, there is no need for the extraction processes to capture and apply changes. Alternatively, where only the latest values are required to be held, the available resources may be deployed more efficiently. Although some might classify this as an implementation consideration, actually it is not. All that is required to be recorded in the conceptual model are the requirements from a business perspective.

The granularity of time is important in data warehousing. It is common in data warehouses for multiple grains of time to be needed within the same model. This might result in more than one time dimension being used and so the requirements for the time granularity must be recorded.

In most of the models under review, the granularity of time is not described in detail and none offer the facility of multiple grains of time. For instance in the RAKE method, as well as others, the data type for time is simply referred-to as a time stamp. There are no provisions for defining more than one domain for time nor for assigning attributes to time domains.

In ERT, time is composed of 'ticks'. It is assumed that the time interval associated with a single tick can vary from application to application. It does appear, however, that multiple grains of time would not be supported as a tick would presumably have the same time interval for a fact as it would for any of the dimensions.

TEER does allow for the adjustment of the granularity of time, although it does not make it clear how the granularity is defined on the model. It does not allow for multiple granularities to be defined on the same model.

Some models have introduced new ideas and new features to show that support for time is needed where, for instance, there is a difference between a 'snapshot' that represents the organisation at a particular point in time but that may change *over time*. As an example, the ERT model adds a 'T' symbol to entities and relationships to show that the values of these objects can change over time. However, in ERT, support for time is limited to the existence of entities and relationships and is not extended to attributes. This implies that row timestamping is an implicit part of ERT. Another model, TERC+, adopts a similar approach but attaches a more novel symbol (🕒) to the affected objects on the diagram.

This approach has the advantage of showing an intuitive model of the organisation while at the same time alerting designers that special considerations must be given to those objects that carry the additional symbols.

RAKE modifies the ER approach by including the identifying attributes for an entity within the entity symbol on the diagram. According to the author, this approach proved useful when modelling history but it is difficult to understand why this would be the case.

The technical designers might feel there is some advantage to be gained in displaying the identifier on the diagram but often they are meaningless to others. In data warehousing, identifiers are often coded and sometimes system generated surrogates. Non-technical people usually have little knowledge of such things and it is hard to see the benefit of including identifiers on the model diagram. In many data warehouse applications, the identifying attributes are hidden from the business users and are used entirely as methods of assigning uniqueness to dimensions. During the transformation processing, the process of format integration may cause the values of some attributes to be altered so that there is consistency in the data warehouse where there is no consistency in the source systems. Experience has shown that this is particularly true with *products* where differing coding systems are used in different parts of the organisation, especially where there are no corporate standards applied to encoding entities. Also in data warehousing the designers, with business approval, will sometimes have to invent codes to provide identifying attributes for dimensional groupings where none existed previously.

The introduction, in the TER model, of the 'OneT' (one at a time) cardinality is a good way of describing the fact that relationship cardinalities change when history is introduced. However, although there are three new relationship cardinalities, only one is of any real value to a data warehouse application, that is: Many-to-OneT – a relationship that has a connectivity of many in one direction and one at a time in the other direction.

The method has some other shortcomings. It does not tackle the issue surrounding the existence of entities and attributes. Instead it focuses attention entirely upon the changing nature of relationships. While this is an important issue in data warehousing, it is by no means the only issue to be dealt with. So this method does not encompass full temporal support at the conceptual level. It would be possible to draw a TER diagram that contained all the temporal requirements of the Wine Club, but each requirement would have to be resolved beforehand and this would mean that the diagram could be regarded as though it implies a

solution. There is a risk that the resulting diagram would be difficult for data warehouse users and other non-technical people to comprehend.

As part of the review of the models, it is worth examining the examples provided by the authors. It is entirely accepted that the principal task of the papers is to explain how the models are constructed. However, in describing the methodologies the examples used are always trivial, some only having two or three entities and none having more than six. Also the nature of the examples is such that there are never any 'event' type entities. The applications tend to model 'relatively static real-life' situations such as Universities or Employee/Employer type scenarios. This approach is good for explaining the principles of the model but none of the methods under review dealt with an event driven example as is typical of a dimensional data warehouse.

It is also important to note that the Wine Club example is quite small for a dimensional model. There are only three dimensions whereas in real life examples there are usually many more. Experience shows that six or more dimensions would be normal. Even so, when using the Wine Club example, some of the diagrams became quite large. Using the standard ER approach or any of the methods under review would tend to complicate the diagram to the point where it becomes very difficult to read.

All the methods under review use symbols to represent relationships in much the same way as the Chen's (1976) ER method. This tendency is probably a reflection of the fact that most of the papers originate from North America where this approach is far more popular than in the United Kingdom. In the representation of relationships, some have utilised the diamond shape, others have used rounded boxes with embedded diamonds. The cardinality and participation conditions are also implemented in various ways: TERC+ opts for one or two lines joining the entity to the relationship to indicate one or many respectively. Another method adopted by RAKE has bars that cross the relationship link lines at right angles. One bar indicates a 'one' relationship while two bars indicates 'many'. The changes that occur to the cardinality of relationships, when time is introduced, are treated in a very explicit fashion in RAKE. The nature of the change from (say) a 'one-to-many' to a 'many-to-many' relationship (from a binary to a ternary relationship) is resolved by the introduction of an intersection entity with time stamps containing the valid time. This reflects, and makes explicit, the traditional

approach to resolving many-to-many relationships and the same approach is adopted for time-varying attributes.

There is a general problem with this approach in that it tends to over-elaborate the depiction of something that is obvious. The purpose of these models is to include the representation of time. It is recognised that the introduction of time variance changes the cardinality of relationships and, effectively, makes them more complex. A temporal data modelling method should accommodate this complexity in an implicit fashion otherwise it cannot be truly classified as a temporal model. The models under review are not significantly more adapted to the representation of time than is the original entity relationship model. In an ER model, the representation of time has to be made explicitly. In most of the models under review, the representation of time, again, has to be made explicit. The only exceptions are TEER and $TEER_2$. These two models assume that all entities, attributes and relationships are temporal. This, clearly, means that they recognise that the representation of time should be more implicit. However, the requirements for a data warehouse are that some of the database objects need to be treated differently, with respect to time, than others.

As a result, to focus on the explicit resolution of such relationships by the use of intersection entities does not add any real value. It does tend to clutter the diagram with repetitious symbols that all serve to represent the same simple property, i.e. time variance.

Another problem with making intersection entities explicit is that there is a risk that it may have the effect of prescribing the final implementation. In a dimensional model, there are several approaches to solving the problem at the logical level. The model does need to be capable of describing the requirements for time but the method for doing so should not be obtrusive so as to dominate the model nor should it imply a specific solution. Designers should be encouraged to consider alternative solutions.

Some consideration should also be given to the subsequent stages of development of the warehouse beyond the conceptual model. Although the conceptual model will be completely independent to the database management systems, there has to be some process by which the conceptual model is allowed to evolve into a logical specification for implementation. As most data warehouses are implemented using a relational data base, it is required that a logical model

can be prepared for a relational database management system[3]. The logical model should encompass all the required temporal components. The requirements surrounding the logical model will be dealt with in the next chapter.

Of the models under review, only one (ERT) supplied a rudimentary mapping from the model to a relational schema. Many of the other models tended to include implementation directives in the conceptual model. For instance, in TEER, the basis of the method is the allocation of a system generated surrogate key value that never changes throughout the lifespan of the entity or relationship. In this respect the method extends into the logical implementation. Also, although TEER claims not to be aligned to the relational model, it comes complete with a query language that is set oriented and has a striking similarity to SQL. With STEER the model strays well into the area of implementation when describing the behaviour of periods and dates. Again, there is also a reference to system generated surrogate keys. The method shares with others the seemingly irresistible urge to specify implementation level solutions that compromise the conceptual model. In TER, the model contains design considerations and compromises due to performance and usage requirements. RAKE goes further and introduces physical indexing and storage constraints into the model.

These issues tend to point to the fact that there is a general confusion as to the purpose of a conceptual model. The definitions of conceptual and logical data models were provided in chapter one. The point is that a conceptual model should describe what is required. It should not prescribe how the requirement should be implemented. It is enough to say that ***this*** data object requires ***that*** kind of support for time without having to be explicit about beginning timestamps and ending timestamps etc. These kinds of considerations should be applied at the logical model level.

The issue of simplicity is worth exploring. Although there is no specific conceptual model for data warehouses, practitioners are generally in agreement that a model of some kind is necessary and also that the model should be easy to understand by the ultimate users of the

[3] The growth in OLAP server products is recognised but the absence of standards makes it impossible to derive a logical schema, other than at a superficial level. [3]

warehouse. Inmon (1992) picks up the point by recommending that the models are created in conjunction with the users. Kimball (1996a) is more forthright and views simplicity as paramount. He goes on to say that simplicity is the central attraction of a dimensional model. He also makes the point that there is a difference between traditional data models that reflect, what he calls, the data dependencies within an organisation and the dimensional model that reflects the business view, as previously described by Codd et al (1993). According to Batini et al (1992), conceptual models should be simple enough for 'users' to understand and contribute to. In terms of notation, there is general acceptance of the fact that diagrammatic representations are easier to comprehend than text. Therefore it is reasonable to specify that one of the components of the model should be a diagram.

It is very difficult to quantify the complexity of a diagram. There is very little in the form of previous research that might provide guidance. James Martin's (1985) view is that end users should be able to read, critique and draw the diagrams quickly. Yourdon (1993) says that it is in the nature of the human mind to only take in a limited amount of information at once and this must be taken into account when constructing ER diagrams. He cites Miller (1956) as evidence of this. Miller's seminal paper on cognitive psychology says that there is evidence to suggest that the human mind can deal with about seven pieces of information at once. Yourdon goes on to state that a diagram should not contain too many components, ideally not more than ten (where each entity and relationship counts as one). Of large ER diagrams he says that there is a large inertia against redrawing the diagram because changing even a small part of it involves considerable overhead. Batini et al (1984) says:

'It is a hopeless matter to define formally what is a pleasant ER diagram and what is not'.

It is not my intention to draw up an evaluation framework for judging the level of complexity of the methods to be reviewed. However, some judgement has to exercised during the review to assess whether each diagram can be regarded as simple enough for non-technical people to use. Following the guidance obtained from the literature, such as it is, I now lay out a set of guidelines.

1. Each dimensional model should comfortably fit on a single page of A4 paper. A dimensional model, in this context, is a single star schema or snowflake schema.

2. The general radial symmetry associated with the dimensional shape must be retained. This means the facts must remain, broadly, near the centre of the diagram and the dimensions and hierarchies should radiate outwards in the typical style of the model.

3. The diagram should lend itself to redrawing. This is to encourage the iterative design approach that is favoured by many developers of methods such as Yourdon (1993) and DeMarco (1979).

4. The number of symbols, and their associated semantics and syntax rules should be minimised.

The issue surrounding the complexity of the models is a major concern in all the examples under review. As the literature says, simplicity and clarity are major requirements of data warehouse models so that non-technical people can comprehend the semantics of the model with little instruction and without having to closely examine the diagram in order to identify the key components. All the methods reviewed were defeated by the purity and simplicity of the basic star format of the dimensional model. The radial nature of the dimensional structure is very quickly lost as the diagrams become more complex. Sometimes it is difficult, even for an experienced practitioner, to pinpoint the facts and distinguish them from the dimensional structures without careful examination.

With the exception of TER, which dealt only with temporal relationships, all the methods attempt to place *all* of the information about the application within the diagram. This includes the entities, relationships, cardinality, participation conditions *and* attributes as well as the temporal variations of each of these. The main problem of trying to include large numbers of objects is that the diagram has to be re-structured to accommodate them. This means shifting the position of entities to make room for other entities and attributes. This is why it becomes difficult to retain the essential shape. It also causes other problems. It was sometimes difficult to restrict a dimensional schema to a single page to the extent that, in many cases, quite a significant amount of time and effort was spent in trying to achieve this objective.

This brings into question the suitability of the diagrams for iterative redrawing. As the diagrams become more complex, the originators are proportionately less likely to make

modifications because the diagrams become increasingly fragile in the sense that even a small change has a significant impact.

The problem of complexity is exacerbated in one model (ERT) where the requirement to include the attributes and the nature of the relationships between entity types and attributes results in a very large number of objects having to be included in the diagram. In order to constrain the Wine Club model to a single page, the size of the symbols has to be reduced and their position has to be chosen carefully. The number of explicit, named relationships is very much larger than that of a traditional ER model. This is largely due to the fact that each attribute in the model participates in an explicit Has/Of relationship with the fact or dimension with which it is associated. This is very repetitious and severely affects the readability of the diagram.

The original dimensional model for the Wine Club, shown in Figure 1.6, contained ten solid symbols (excluding connecting lines, degree and participation conditions). The modified entity relationship model shown in 3.1 contained fifteen solid symbols and this was felt to be rather complex for such a simple example. The average number of solid symbols in the temporal models excluding TER, which only dealt with relationships, is sixty-one. The model with the maximum number of solid symbols is ERT, with ninety-one.

3.11 Conclusion

Insofar as the representation of time is concerned, there are differing views about the temporal nature of the three main components of the models (entities, relationships and attributes) and how they should be treated. In fact I have concluded that they should all be treated in the same way. This can be reasoned as follows:

The temporal requirement for an entity relates to its existence. The existence of an entity may, in some circumstances, be discontinuous. For instance a Wine may be sold by the Wine Club for a period of time and then be withdrawn. Thereafter it may again come into existence for future periods of time. This discontinuity in the life cycle of the wine may be important when queries such as 'How many product lines do we sell currently?' are asked. A further example concerns customers. The regaining of lost customers is increasingly becoming an important business objective especially in the telecommunication and financial services industries. When

customers are 'won back', the companies prefer to reinstate them with their previous history intact. It is now becoming more common for customers to have discontinuous existences.

A relationship that requires temporal support can be modelled as a m:n relationship. Using the ER standard method for resolving m:n relationships, a further entity, an intersection entity, would be placed between the related entities. So there now exists another entity (a weak entity). As the temporal requirement for an entity relates to the existence of the entity, the treatment of relationships is precisely the same as the treatment of entities. It is a question of existence once again.

An attribute should really be regarded as a property of an entity. The entity is engaged in a relationship with a domain of values. Over time, that relationship would necessarily be modelled as a m:n relationship. An attribute that requires temporal support, therefore, would be resolved into another intersection entity between the entity and the domain, although the domain is not actually shown on the diagram. The treatment of attributes is, therefore, the same as entities and relationships and relates to the existence of the value of the attribute for a period of time.

Turning to the representation of time, the models under review tended to opt for one of two approaches:

The most popular approach is to include the attributes on the diagram as this provides a complete picture of the requirements with respect to time. However, in all but the most trivial example, the picture rapidly becomes cluttered with objects with the effect that it quickly becomes difficult to understand.

Dimensions consisting of large numbers of attributes are regarded as common. According to Kimball (1996a):

> ***'A good rule of thumb for a serious product dimension for a large manufacturer or retail company is that there should be at least 50 descriptive text fields, and 100 is not a foolish number'. (page 92)***

Not all of the dimensions in a model would have large numbers of attributes but even if just one or two dimensions are quite large, the affect on the diagram would be that it may become,

potentially, quite difficult to read. However, the omission of attributes from the model would be confusing as only part of picture, with respect to time, is then visible. It makes the model easier to read at a superficial level but it requires the inclusion of supporting documentation to incorporate the attributes and their requirements with respect to the treatment of time. This would present an inconsistent picture.

The second option, chosen by TEER and $TEER_2$, is to assume that all the objects on the model are temporal and, therefore, the diagram remains unchanged. This is not a realistic approach to be adopted for a data warehouse where, at the present time, choices have to be made as to the individual temporal requirements for each component of the model.

It seems that an 'all or nothing' approach provides the most consistent and least confusing solution. Either all the temporal requirements must be shown on the diagram or none.

In view of the issue of complexity, I have concluded that the best approach is to omit the temporal information from the diagram so that it can be included fully on supporting documentation. There will always be a need for supporting information to convey details of such things as the metadata, constraints and assumptions. By transferring some of the information to supporting documents the main diagram is more likely to be able to retain its essential shape and would be easier to understand.

The question that was raised at the beginning of this chapter, whether or not a single model was capable of enabling the requirements of both the business and technical designers to be met, can now be answered. The models under review are all usable by technical designers in that they can describe the formal requirements that are needed to produce a logical model. However, they do not satisfy the requirements of the business people, as described in the previous section. It would not be feasible to expect business people to be able to construct and validate these models.

What is needed is a higher level of abstraction that enables the separation of the capture of the 'informal' business requirements so that business people can readily build and validate that part of the model. The requirements for the representation of time of each data object, in business terms, can be captured on another part of the model so that the designers can go on to develop a logical model.

4 Logical models for time in data warehousing

This chapter of the thesis describes some solutions to the problems of the representation of time in data warehousing, as recommended by both academics and practitioners. In particular, attention is focused on the representation of time in the dimensional structures of a dimensional data warehouse. The representation of time in the fact table was covered in section 2.2.1.

Logical models systematically follow conceptual models and so a natural consequence of the failure to define the requirements in the conceptual model is that the requirements are also absent from the logical and physical implementation. As a result, practitioners have subsequently found themselves faced with problems involving time and some have created solutions. However, the solutions have been developed on a somewhat ad-hoc basis and are by no means comprehensive. Also, as has been stated previously, there has been very little academic research carried out in this field. Only one other dissertation has been found covering data warehousing and it is not at all connected with the subject of this research.

A comprehensive literature search revealed many magazine articles, mostly in trade journals, on the subject but many of these are product introductions or advertisements, sometimes in the guise of 'white papers'. A previous investigation by Sakaguchi and Frolick (1996) came to a similar conclusion. Many of the articles deal with data warehousing at a very superficial level and very few indeed, could be described as deeply technical. The literature review covers both the conceptual and logical levels of data warehouse design.

The solutions that have been developed by practitioners in the field of data warehousing are reviewed to assess their ability to solve the problems described in chapter two. Some of the techniques have been reviewed by academics but, again, there has been very little research on this subject so far. The principal method is one invented by Ralph Kimball (1996a) that is described in detail in the next section. Where others have made contributions, these are also described. Some authors, such as Inmon et al (1997) make mention of the subject but are too vague or too superficial in their approach to a resolution. These documents are not reviewed.

Others have put forward data warehouse development methodologies. These include Menninger (1995b), Griffin (1997), Fererra and Naecker (1993) and Bohn (1997). The solutions tend to be logical or physical in their description and are invariably based on the relational model. Their treatment of time is too superficial to be reviewed.

In developing his solution, Kimball (1996a) recognises that

> ***'One of the main responsibilities of a data warehouse is to correctly represent prior history status'. (Page 100)***

He then claims that most dimensions of analysis in a dimensional schema remain almost constant over time and that we should exploit this fact in our model. However, in what appears to be a contradictory statement in the same book, he says that we cannot simply assume that dimensional attributes remain constant over time. Customers, he concedes, are constantly changing. This last statement certainly appears to be true. There is evidence in a housing survey by Green et al (1997) that, in the United Kingdom, people change their addresses, on average, every ten years. This means that an organisation can expect to have to implement address details changes to about ten percent of their customers each year. Over a ten year period, if an organisation has one million customers, it can expect to have to deal with one million changes of address. Obviously, some people will not move but others will move more than once in that period. This only covers address changes. There are other attributes relating to customers that will also change, although perhaps not with the same frequency as addresses.

As far as implementation is concerned, there are differing views as to whether the logical implementation should be a star schema or a snowflake. Where performance is concerned, it is generally accepted that queries are more efficient when implemented using a star. For instance, queries that involve joins between the fact table and dimensions will be more efficient where the dimensions are not organised in the fashion of a snowflake as this, inevitably, involves more physical joins. Kimball's main reason for rejection of snowflakes is the potential for the degradation in performance due to the necessity for joins to be executed in dimensional browsing queries. He says, as a design principle:

'Do not snowflake your dimensions, even if they are large.
If you do, be prepared to live with poor browsing performance'. (page 97)

His claim is that the only reason for creating dimensional hierarchies is to save disk storage space and that the amount of disk storage space saved is insignificant when considering the data warehouse as a whole.

As his main concern is for performance, Kimball's solution to dimensional hierarchies is to 'flatten' then into a single (i.e. denormalised) table.

The conceptual model does not necessarily have to be logically implemented in a 1:1 Entity→Table basis. So the diagram does not always reflect the physical table structure. It is perfectly feasible to present a snowflake structure to business people and to implement a perfect star schema at the logical level, if that is the best way to do it from a usability and performance perspective. Poe et al (1997) recognise that, although users often wish to see the hierarchies, a well designed star schema should be able to encompass them within the dimension tables. As far as performance is concerned, Morse and Isaac (1998) agree entirely with Kimball and say that the major advantage of a star schema is that it supports the goal of very high sustained rates of data I/O.

Morse and Isaac's view is that, as a general rule in data warehouses, normalised schemas should be avoided if performance is to be acceptable. The relational database vendor, Oracle (Dodge and Gorman (1998)), implies that the true star is preferable to the snowflake without actually saying so. It says that natural dimensional hierarchies *can be* 'collapsed' into single dimensions. Dodge and Gorman also state that, in the future, data warehouses will be implemented using star schemas because they are:

'more flexible and save a lot of space and advance processing effort'. (page 142)

Another relational database vendor, Sybase (Hammergen (1998)), does not actually express a preference but all the examples in the book are true star schemas.

These are physical and performance considerations that are not the subject of this thesis. However, from a practitioner's perspective, the problem can be resolved on a 'per case' basis. A logical star schema can be presented to the users as a conceptual snowflake and vice versa,

depending on the preferences of the organisation concerned. The simplest way to do this is by the implementation of userchemas, or 'views' in the case of a relational database.

4.1 Review of Kimball's approach

I will now describe the method that Kimball recommends for the representation of time in data warehouses.

His collective term for changes to dimensional attributes is 'Slowly Changing Dimensions'. The term has become well known within the data warehouse industry and has been adopted generally by practitioners. He cites three methods of tracking changes to dimensional attributes with respect to time. These he calls simply types One, Two and Three.

Within the industry, practitioners are generally aware of the three types and, where any support for time is provided in dimensional models, these are the approaches that are normally used. It is common to refer to products and methods as being consistent with Kimball's type one, two or three. In a later work Kimball (1998a) recognises a 'type zero' that represents dimensions that are not subject to change.

4.1.1 The 'type one' approach

The first type of change, known as 'type one' is to replace the old data values with the new values. This means that there is no need to preserve the previous value. The advantage of this approach, from a system perspective, is that it is very easy to implement. Kimball recognises that this method does not achieve the real goal of faithfully tracking history. Obviously there is no temporal support being offered in this solution. He describes it as *'not always crazy'* as it is the only effective way of handling the correction of errors. My view is that, far from being 'crazy', this method is sometimes the correct method to be used. In the Wine Club example, attributes like the Member's name or the member's main hobby are best treated in this way. These are attributes for which there is no requirement to retain historical values. Only the latest value is deemed by the organisation to be useful. All data warehouse applications will have some attributes where the correct approach is to overwrite the previous values with the new values, quite apart from correcting errors.

It is important that the updating is effected on a per attribute basis rather than a per row basis. Each dimension will have a mixture of attributes, some of which will require the type one replacement approach while others will require a more sophisticated approach to the treatment of value changes over time.

The worst scenario is a full table replacement approach where the dimension is, periodically, completely overwritten. The danger here is that any rows that have been deleted in the operational system may be deleted in the data warehouse. Any rows in the fact table that refer to rows in the dimension that have been deleted will cause a referential integrity violation and will place the database in an invalid state.

Thus, the periodic update of dimensions in the data warehouse must involve only inserts and updates. Any logical deletions (e.g. where a customer ceases to be a customer) must be processed as updates in the data warehouse. It is important to know whether a customer still exists as a customer but the customer record must remain in the database for the whole lifespan of the data warehouse or, at the very least, as long as there are fact table records that refer to the dimensional record.

Due to the fact that type one is the simplest approach, it is often used as the default approach. Practitioners will sometimes adopt a 'type one' solution as a short term expedient, where the application really requires a more complete solution, with the intention of providing proper support for time at a later stage in the project. Too often, the pressures of project budgets and implementation deadlines force changes to the scope of projects and the enhancements are abandoned. Sometimes, type one is adopted due to inadequate analysis of the requirements with respect to time.

4.1.2 The 'type two' approach

4.1.2.1 General review

The second of Kimball's solutions to slowly changing dimensions is called type two. Type two is a more complex solution than type one and does attempt to faithfully record historical values of attributes by providing a form of version control. Type two changes are best explained with the use of an example. In the case study, the vintage (year of production) of a bottle of wine is subject to change. There is a requirement to faithfully reflect popularity over

time. This means that the vintage prevailing at the time of the sale must be used when analysing sales. If the type one approach was used when recording changes to the vintage, the historical values will appear to have the same vintage as current values. A method is needed, therefore, which enables us to reflect history faithfully.

The type two method attempts to solve this problem by the creation of new records. Every time an attribute's value changes, if faithful recording of history is required, an entirely new record is created with all the unaffected attributes unchanged. Only the affected attribute is changed to reflect its new value. The obvious problem with this approach is that it would immediately compromise the uniqueness property of the primary key, as the new record would have the same key as the previous record. Kimball (1996g) uses this to advantage and introduces the use of surrogate keys. Kimball prefers to use surrogate keys in all the dimensions of analysis. The main reason for this is that the production key is subject to change whenever the company reorganises its customers or products and that this would cause unacceptable disruption to the data warehouse if the change had to be carried through. It is better to create an arbitrary key, to provide the property of uniqueness. So each time a new record is created, following a change to the value of an attribute, a new surrogate key is assigned to the record. Kimball uses the term 'generalised keys', in place of surrogates, as his design principle states:

> ***'The use of the type two slowly changing dimension requires that the dimension key be generalised. It may be sufficient to take the underlying production key and add two or three version digits to the end of the key to simplify the key generation process'. (Page 102)***

He offers two approaches to assigning the value of the surrogate:

1. The identifier is lengthened by a number of version digits. So a Wine having an identifier of '1234' would subsequently have the identifier '1234001'. After the first change, a new row would be created that would have an identifier of '1234002'. The wine would now have two records in the dimension. Most of the attribute values would be the same. Only the attribute, or attributes, that had changed would be different.

2. The identifier could be truly generalised and bear no relation to the previous identifiers for the Wine. So each time a new row is added, a completely new identifier is created. According to Kimball, the best way to do this is to use a sequentially assigned integer and says that it is best to simply start with 1 and continue up to the highest number that is needed.

The generalised key is used in both the dimension table and the fact table. Constraining queries using a descriptive attribute, such as the name of the wine, will result in all records for the wine being retrieved. Constraining or grouping the results by use of the name and, say, the 'Vintage' attribute will ensure that history is faithfully reflected in the results of queries.

The type two approach, therefore, will ensure that the fact table will be correctly joined to the dimension and the correct dimensional attributes will be associated with each fact. Ensuring that the facts are matched to the correct dimensional attributes is Kimball's main concern.

Another variation of this approach, which Kimball has not identified, is to use a composite identifier could be constructed by retaining the previous number '1234' and adding a new 'Version' attribute which, initially, would be '1'. This approach is similar to 1, above. The initial identifier is allocated when the Wine is first entered into the database. Subsequent changes require the allocation of new identifiers. According to Kimball, it is the responsibility of the Data Warehouse administrator to control the allocation of identifiers and to maintain the version number in order to know what version number, or generalised key, to allocate next. This modified approach to type two is described by Kamfonas (1998).

In reality, the requirement would be to combine type one and type two solutions in the same logical row. This is where we have some attributes that we do want to track and some that we do not. An example of this occurs in the Wine Club where, in the Member dimension, we wish to trace the history of attributes like the address and, consequently, the sales area but we are not interested in the history of the member's name or their hobby code. So in a single logical row, an attribute like address would need to be treated as a type two whereas the name would be treated as a type one.

Therefore, if the member's hobby changes, we would wish to overwrite it. However, there may be many records in existence for this member, due to previous changes to other

attributes. Do we have to go back and overwrite the previous records ? Kimball does not examine this issue. In practice, it is likely that only the latest record would be updated. This implies that, in dimensions where type two is implemented, attributes for which the type one approach would be preferred will be forced to adopt an approach that is nearly, but not quite, type two.

In chapter two, describing the problems surrounding 'time' in data warehousing, I showed how the results of a query could change due to a member moving house. The approach taken was to simply overwrite the old addresses and the sales area codes with the new values. This is equivalent to Kimball's type one approach.

We will now implement the same changes using the type two method

Using Kimball's type two method we create new rows for the changing members with generalised keys, as shown in Figure 4.1:

```
Insert into member
        select 8876, member_name, member_address,
        sales_area_code, hobby_code, date_joined
        where member_code = 1136
Insert into member
        select 8877, member_name, member_address,
        sales_area_code, hobby_code, date_joined
        where member_code = 1651
Insert into member
        select 8878, member_name, member_address,
        sales_area_code, hobby_code, date_joined
        where member_code = 1404
Update member set
          member_address='44 Sea View Terrace, West Bay,
               Bridport, Dorset', sales_area_code='SW'
        where member_code = 8876
Update member set
          member_address='Bay View Cottage, The Promenade,
             Weymouth, Dorset', sales_area_code='SW'
        where member_code = 8877
Update member set
          member_address='Flat 10, The Heights, Cherry Road,
             Torbay, Devon', sales_area_code='SW'
        where member_code = 8878
```

Figure 4.1 Updating dimensions using the Type Two method

The next available generalised key is assumed to be 8876 and we have further assumed that the keys are in strict ascending sequence.

We now have two records for each of the members who have moved house.

If we now re-run the original query from Figure 2.12, the result set in Figure 4.2 is returned

Sales Area	Bottles	Revenue
North East	9628	65083.05
North West	4077	29329.06
South East	5662	39597.12
South West	5907	41560.58

Figure 4.2 Total quantity and revenue by sales area

This is the same as the original result shown in Figure 2.13. Thus I conclude that the analysis has not been disrupted by the changes in address nor the consequent causal changes in Sales Area.

Future insertions to the Sales fact table will be related to the new identifying Member Codes and so the segmentation will remain consistent with respect to time for the purposes of this particular query.

One potential issue here is that, by making use of generalised keys, it becomes impossible to recognise individual members by use of the identifying attribute. As each subsequent change occurs, a new row is inserted that is identified by a key value that is in no way associated with the previous key value. In the example above, one member's original value for the Member Code attribute was 1136 whereas the value for Member Code for the new inserted tuple was 8876, which was the next available key in the domain range.

This means that, if it was required to extract information on a 'per member' basis, the grouping would have to be on a non-identifying attribute, such as the member's name i.e.:

```
select member_name "Name",
       sum(quantity) "Bottles",sum(value) "Revenue"
       from sales s, member c
       where c.member_code=s.member_code
       group by member_name
       order by member_name
```

Figure 4.3 Query grouping using descriptive attributes

Constraining and grouping queries using descriptive attributes like names is clearly risky since names are apt to be duplicated and erroneous results could be produced.

Another potential issue with this approach is that, if the keys are truly generalised, as with key hashing, it may not be possible to identify the latest record by simply selecting the highest key. Also, the use of generalised keys means that obtaining the history of, say, a member's details may not be as simple as ordering the keys into ascending sequence.

One solution to this problem is the addition of a constant descriptive attribute, such as the original production key, that is unique to the logical row. Alternatively, the use of Kamfonas (1998) variation, as previously described, in which the original key is retained but is augmented by an additional attribute to define the version, would also provide the solution to this.

Kimball rejects the use of date columns to identify when changes actually take place. For instance, this means that it is not possible to establish with any accuracy when a member actually moved house. The only date available to provide any clue to this is the transaction date in the fact table. There are some problems associated with this. A query such as *"List the names and addresses of all members who have purchased more than twelve bottles of wine in the last three months"* might be useful for campaign purposes. Such a query will, however, result in incorrect addresses being returned for those members who have moved house but not since placed an order. The query in Figure 4.4 shows this:

```
select c.member_code, member_name, member_address,sum(quantity)
    from member c, sales s, time t
    where c.member_code = s.member_code
    and s.time_code = t.time_code
    and t.month in (199810, 199811, 199812)
    group by c.member_code,member_name,member_address
    having sum(quantity) > 12
```

Figure 4.4 Query to produce a campaign list

The table in Figure 4.5 is a subset of the result set for the query in Figure 4.4:

Member Code	Member Name	Member Address	
1332	A.J.Gordon	82 Milton Ave,Chester,Cheshire	49
1315	P. Chamberlain	11a Mount Pleasant,Sunderland	34
2131	Q.E.McCallum	32 College Ride,Minehead,Somerset	14
1531	C.D.Jones	71 Queensway,Leeds,Yorks	31
1136	A.G.Andrew	9Broughton Hall Ave, Woking,Surrey	32
2141	J.K Noble	79 Priors Croft,Torquay,Devon	58
4153	C.Smallpiece	58 Ballard Road,Bristol	21
1321	D.Hartley	88 Ballantyne Road,Minehead,Somerset	66

Figure 4.5 List of members to be contacted

The highlighted row is an example of the point. Member 'A.G.Andrew' has two entries in the database. The change of address was effected in a previous exercise (see Figure 4.1). This member has not purchased any wine since moving house and, therefore, the incorrect address was returned by the query. The result of a simple search is shown in Figure 4.6.

Member Code	Member Name	Member Address
1136	A.G.Andrew	9Broughton Hall Ave, Woking,Surrey
8876	A.G.Andrew	44 Sea View Terrace, West Bay, Bridport, Dorset

Figure 4.6 Multiple records for a single entity

If it can be assumed that the generalised key is always ascending, then the query could be modified, as in Figure 4.7, to select the highest value for the key:

```
Select member_code, member_name, member_address
    from member
    where member_code = (select max(member_code)
    from member
    where member_name = 'A.G.Andrew')
```

Figure 4.7 Obtaining the current address

This query returns the second of the two rows listed in Figure 4.6.

Using Kimball's other technique to implement the type two method, we could have altered the Member code from '1136' to '113601' for the original row and, subsequently, to '113602' for the new row containing the changed address and sales area.

In order to return the correct addresses, the following query has to be executed:

```
select c1.member_code,member_name, member_address,sum(quantity)
    from member c1, sales s, time t
    where c1.member_code = s.member_code
    and c.member_code = (select max(member_code)
                from member c2
                where substr(c1.member_code,1,4)
                    = substr(c2.member_code,1,4))
    and s.time_code = t.time_code
    and t.month in (199810, 199811, 199812)
    group by c1.member_code,member_name,member_address
    having sum(quantity) > 12
```

Figure 4.8 Obtaining the latest members details using type two with extended identifiers

The query in Figure 4.8 is another correlated sub-query and contains the following line:

```
where substr(c1.member_code,1,4) = substr(c2.member_code,1,4)
```

The query matches the generic parts of the member code by use of a 'substring' function, provided by the query processor. It is suspected that this type of query may be beyond the capability of most users.

This approach is dependant on the fact that all the codes are of the same fundamental format. That is, they have four digits plus a suffix. If the approach was to start from 1 through 9999, then this technique could not be adopted because the substring function would not produce the right answer.

The obvious variation on the above approach is to add an extra attribute to distinguish versions. The identifier then becomes the composite of two attributes instead of a single attribute. In this case, the original attribute remains unaltered, and the new attribute is incremented, as shown in figure 4.9:

Member Code	Version Number
1136	01
1136	02
1136	03

Figure 4.9 A modification to type two using composite identifiers

This modification to Kimball's method is another example of Kamfonas (1998) variation. He refers to it as 'direct versioning'.

Using this technique, the following query is executed:

```
select c1.member_code,member_name, member_address,sum(quantity)
    from member c1, sales s, time t
    where c1.member_code = s.member_code
    and s.counter = c1.counter
    and c.counter =
    (select max(counter) from member c2 where c1.member_code =
    c2.member_code)
    and s.time_code = t.time_code
    and t.month in (9810,9811,9812)
    group by c1.member_code,member_name,member_address
    having sum(quantity) > 12
```

Figure 4.10 Obtaining the latest members details using type two with composite identifiers

The structure of this query is the same as in Figure 4.8. However, this approach does not require the use of substrings to make the comparison. This means that the query will always produce the right answer irrespective of the consistency, or otherwise, of the encoding procedures within the organisation.

These solutions do not resolve the problem of pinpointing *when* a change occurs. Due to the absence of dates in the type two method, it is impossible to determine precisely when changes occur. The only way to extract any form of alignment with time is via a join to the fact table. This, at best, will give an approximate time for the change. The degree of accuracy is

dependent on the frequency of fact table entries relating to the dimensional entry concerned. The more frequent the entries in the fact table, the more accurate will be the traceability of the history of the dimension and vice versa. It is also not possible to record gaps in the existence of dimensional entries. For instance, in order to track precisely the discontinuous existence of, say, a customer there must be some kind of temporal reference to record the periods of existence.

4.1.2.2 Dimensional hierarchies

Kimball's approach to the structure of data warehouses is now examined in detail. The approach is founded on the star schema. In this sense we refer to a true star schema where there is a single fact table and two or more dimension tables attached to it. Kimball (1996a) proscribes the use of snowflake schemas for two reasons. The first is the effect on performance that has already been described. The second reason is as follows:

> ***'The average user is intimidated by this (snowflake) diagram. All other technical considerations aside, this impact on our end users is enough to recommend against using a snowflake structure to represent hierarchies'. (Page 95)***

My experience has shown his assertion, that users and business people do not understand hierarchies, to be quite untrue. My experience is, in fact, the opposite. Most business people are very aware of hierarchies and are confused when you leave them out or try to flatten them into a single level.

Kimball uses the hierarchy in Figure 4.11 as his example:

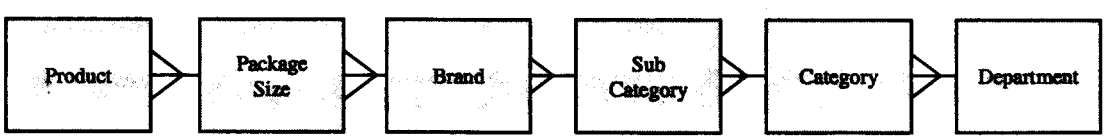


Figure 4.11 Kimball's example of a business hierarchy

It is possible that it is this seven layer example that is confusing and intimidating, rather than the principle. In practice, hierarchies involving seven layers are almost unheard of and this example, it is assumed, has been introduced for reasons of hyperbole. It is difficult to obtain opinions from others in the field although Raden (1996) states that dimensional modelling

always involves hierarchies in dimensions and that using a snowflake schema approach makes sense with large dimensions, although he does not say why this is.

Kimball (1998a and 1998c) is prepared to implement snowflake schemas, albeit reluctantly, to ease certain design issues. These examples are not relevant to this investigation and will not be discussed further.

Far more common are hierarchies like this one reproduced from the Wine Club case study, shown in Figure 4.12.

Figure 4.12 Simple business hierarchy

These hierarchies are very natural to managers because, in real life scenarios, Customers (Members) are organised into geographical regions or market segments and the managers' desire is to be able to ask questions about the business performance of this type of segmentation. Similarly, managers are quite used to comparing the performance of one department against other departments, or one product range against other product ranges.

The whole of the business world is organised in a hierarchical fashion. We live in a structured world. It is when we try to remove it, or flatten it, that business people become perplexed.

So I would offer a counter principle with respect to snowflake schemas:

There is no such thing as a true star schema in the eyes of business people. They expect to see the hierarchies.

In the review of Kimball's (1996g) methods, we have to keep in mind that his view of the logical implementation is a perfect star schema. The conceptual dimensional hierarchies, that are acknowledged by Kimball, have been physically denormalised into single level dimensions.

4.1.2.3 Problems with hierarchies

So far in this chapter, attention has focused on so-called 'Slowly Changing Dimensions' and how these might be supported using the type two solution. Now we will turn our attention to, what we shall call, 'Slowly Changing Hierarchies'. As an example, I will use the dimensional hierarchy illustrated in Figure 4.12. The attributes, using a surrogate key approach, are as follows:

Sales_Area(Sales_Area_Key, Sales Area Code, Manager key, Sales Area Name)

Manager(Manager_Key, Manager Code, Manager Name)

Member(Member_Key, Member code, Member Name, Member Address, Sales Area key, Hobby Code,Date Joined)

The number of members and the spread of sales areas in the case study database is as shown in the table in Figure 4.13:

NE	20
SW	15
NW	8
SE	9

Figure 4.13 Members grouped by sales area

We will assume that we have implemented the type two solution to slowly changing dimensions.

If sales area 'SW' was to experience a change of managers from M9 to M12, then a new sales area record would be inserted with an incremented Counter, together with the new manager_code. So if the previous record was: (1, SW, M9, 'South West'), the new record with its new key, assumed to be 5, would contain (5, SW, M12, 'South West').

However, that is not the end of the change. Each of the Members, from the 'SW' sales area still have their foreign key relationship references pointing to the original sales area record containing the reference to the old manager (Surrogate key 1). Therefore, we also have to create an entire set of new records for the members, with each of their sales area key values set

to '5'. In this case, there are fifteen new records to be created. It is not valid to simply update the foreign keys with the new value because the old historical link will be lost.

Where there are complex hierarchies involving more levels and more rows, it is not difficult to imagine very large volumes of inserts being generated. For example, in a four level hierarchy where the relationship is just 1:100 in each level, a single change at the top level will cause over a million new records to be inserted. A relationship of 1:100 is not inordinately high when there are many data warehouses in existence with several million customers in the customer dimension alone.

The issue surrounding slowly changing hierarchies is not addressed by Kimball in his approach to the design of dimensions. He is concerned about performance. The number of extraneous insertions generated by this approach could cause the dimension tables to grow at a rate which, in time, becomes a threat to performance.

Kimball advocates flattening the hierarchies into a single dimension (de-normalising). This converts a snowflake schema into a star schema. If this approach is taken, the effect is that, in the Wine Club example, one insertion is saved. In the four level 1:100 example, the number of insertions reduces from 1.01 million insertions to 1 million insertions. So reducing the number of insertions is not a reason for flattening the hierarchy.

4.1.2.4 Dimension browsing.

The type two approach does cause some problems when it comes to dimension browsing. According to Kimball, some eighty percent of data warehouse queries are dimension browsing queries. This means that they do not access the fact table at all.

A typical dimensional browse query we would wish to perform is to count the number of occurrences. For instance, how many customers do we have? The standard way to do this in SQL is shown Figure 4.14:

```
Select count(*)
        from <table>
        where <predicate>
```

Figure 4.14 Simple general query to count members

In a type two scenario, this will produce the wrong result. This is because for each logical record, there are many physical records that result in a number of 'duplicated' rows.

Take the example of a sales exec entity and a customer entity shown in Figure 4.15:

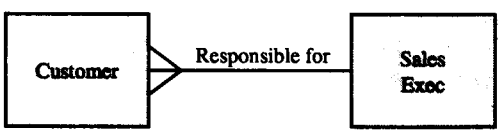


Figure 4.15 Simple general business hierarchy

In order to count the number of customers that a sales exec is responsible for, a non-expert user might express the query as shown in Figure 4.16

```
Select count(*)
        from Sales_Exec S,Customer C
        where S.SalesExecNum=C. SalesExecNum
        And S.Name = 'Tom Sawyer'
```

Figure 4.16 Non expert query to count customers

When using the type two approach to allow for changes to a customer attribute, this will produce the wrong result. This is because for each customer there may be many rows, resulting from the duplicates created when an attribute value changes. With more careful consideration, it would seem that the query should instead be expressed as shown in Figure 4.17:

```
Select count(distinct <logical primary key>)
        from <table>
        where <predicate>
```

Figure 4.17 Simple general query to count customers, eliminating duplicates

In the example that translates to the query in Figure 4.18:

```
Select count(distinct CustNum)
        from Sales_Exec S,Customer C
        where S. SalesExecNum =C. SalesExecNum
        and S.Name = 'Tom Sawyer'
```

Figure 4.18 Non expert query to eliminate duplicates in a Type Two scenario

Unfortunately, the query in Figure 4.18 does not give the correct result either because the result set contains all the customers that this sales executive has ever been responsible for at any time in the past. The query includes the customers that are no longer the responsibility of Tom Sawyer. When comparing sales executives, this would result in customers being double counted. On further examination, it might appear that this problem can be resolved by selecting the latest entry for each customer to ensure that they are counted as a customer for only their current sales executive. Assuming an incremented general key, this can be expressed by the query in Figure 4.19:

```
Select count(*)
        from Sales_Exec S,Customer C1
        where S. SalesExecNum=C1. SalesExecNum
        And S.Name = 'Tom Sawyer'
        and C1.Counter =
                (Select max(Counter from Customer C2)
                where C1.CustNum=C2.CustNum)
```

Figure 4.19 Attempting to find the latest occurrence of customer details

In fact, the query in Figure 4.19 also gives invalid results because it still includes customers that are no longer the responsibility of Tom Sawyer. That is, the count includes customers who are not currently the responsibility of any sales executive, because they are no longer customers, but whose records must remain because they are referenced by rows of the fact table.

This example should be seen as typical of data warehouses, and the problem just described is general in this situation. That is, using Kimball's type two approach, there are simple dimensional queries that cannot be answered correctly. I conclude that the type two approach does not fully support time in the dimensions. Its purpose, simply, is to ensure that fact entries are related to the correct dimensional entries such that each fact, when joined to any dimension, displays the correct set of attribute values.

4.1.2.5 The 'supplies from stock' problem.

Kimball considers that, when the value of a dimensional attribute changes, the time of the change is not important and says:

> *'There is no need to place an effective date for the change'. (Page 103)*

He is concerned that the effective date might be used as a constraint in queries. He considers this to be so important that he introduces the following design principle:

> *'The Type Two slowly changing dimension automatically partitions (provides a historical reflection of) history and an application must not be required to place any time constraints on effective dates in the dimension'. (Page 103)*

Kimball's reasoning is associated with the 'supplies from stock' problem described in the previous chapter. He is concerned that queries will join individual sales events to the wrong dimensional record if the join is effected by use of the sales event date and some 'effective date' on the dimension record. He goes on to say:

'Although we could put an Effective Date in the product dimension record, we cannot use it as a constraint to partition (provide a historical reflection of) sales because it has no physical relevance and would give us the wrong answer'. (Page 103/4)

It is worth exploring this a little more using Kimball's (1996a) own example. I have reproduced the text as printed:

> *In the case of cans of soup, we imagine that a product dimension represents a particular universal product code (UPC). At some point in time, say January 15th 1994, the salt content formulation for newly manufactured cans of soup changes from salty to salt free, reflecting a real change in the soup's ingredients. In real grocery stores, this situation occurs frequently and the grocery industry routinely issues a new UPC code to describe the altered product. We see that, in this context, the UPC code is nothing more than a glorified generalised key maintained by the manufacturer of the soup.*
>
> *Now, when we look at the database of grocery store sales, we don't see such an abrupt partitioning of history. The old cans of soup will continue to sell in stores past January 15th 1994 until they are depleted. This will vary from store to store. The new soup will appear on the shelves no earlier than January 15th 1994 and will gradually supersede the old soup. There will be a transition period where, in any given store, both types of cans will be moving past the cash registers. The cash registers will recognise both UPC codes and will have no difficulty handling the sale of either type of soup. Again, the use of separate dimension records automatically partitions the fact table and the user doesn't need to be concerned with whether there are two formulations of the soup unless the user is using the salt content field. The important point is that although we could put a manufacturing effective date in the product dimension record, we don't dare constrain on it to partition sales because it has no physical relevance and would give us the wrong answer.*

So it appears that Kimball is not dismissing the idea of recording the times of changes in attribute values, he is simply saying that, because he feels they cannot be used as constraints in querying the facts, they are irrelevant. This tends to emphasise the fact that Kimball regards the dimensions merely as extensions of the facts. However, it is important to examine his reasoning closely to determine whether or not this is a general problem.

The type two solution to slowly changing dimensions is based on the creation of new records and the allocation of surrogate keys when changes are detected in dimensional attributes. Usually, it is the task of the data warehouse to detect, or to be informed, that a change has occurred. The warehouse processing is responsible for the maintenance and allocation of the surrogate key. Kimball is quite clear about this and it is the most sensible approach. It is unrealistic to expect the operational systems to maintain the surrogate key system on behalf of the data warehouse.

Let us now assume that a sale has occurred, of a can of soup. The data warehouse loading process would routinely attempt to add the appropriate foreign key for the product dimension to the sale record. Immediately we have a problem because the warehouse process would have no way of knowing whether the can of soup is of the salty or salt free variety and would not know how to deal with it. The only possible way of dealing with the problem successfully would be if the operational system had provided the product code. In the example provided by Kimball, that is precisely what happened. The manufacturer created an entirely new product with its own unique UPC. The data warehouse processing would detect that a new product had been created and would duly create its own version of the new product in its product dimension. The sales record, when it appeared, would already be bearing the new UPC. Kimball has compared the UPC as a kind of surrogate key and that may be correct. However, this is not an example of type two. It is the manufacturer that assigned the UPC, not the data warehouse. As far as the data warehouse is concerned, this is a new product and not a variation of an existing product and its type two processing would not be invoked. The type two approach, where the data warehouse is responsible for maintaining and assigning the surrogate keys, cannot be used in the scenario that Kimball describes.

The logical way for type two to be used is as follows: for each sales record to be inserted into the fact table, the warehouse processing would have to match the product code attached to the

sales record against some mapping table that would yield the latest surrogate key that had been allocated to the product. A more sophisticated approach involving the additional matching of the sales date to a product period is not appropriate for the reason Kimball has explained. A separate process would be responsible for detecting changes to dimensional attributes. When a change is detected, a new record would be created with a new surrogate key and the mapping table would be altered to reflect the new situation. Any future sales of the product would be assumed to be related to the latest version of the surrogate key.

Therefore, the type two solution is no more effective in resolving the 'supplies from stock' problem than an approach that uses dates. Kimball's assertion that the use of dates has *'no physical relevance and would give us the wrong answer'* applies equally to constraints involving surrogate keys. The problem can only be resolved by the operational system supplying the data warehouse with sufficient information to enable the correct assignment to take place. The problem is really one of stock control, rather than a data warehousing or a temporal issue.

Although this problem is presented by Kimball as a major issue, it only applies in a minority of cases. Firstly, it affects only the 'product' dimension. Other dimensions such as customers, geography, distribution and marketing classifications would not be similarly affected. Secondly, it only applies in retail applications. Others such as telecommunications, banking, insurance and utilities are not affected by this.

4.1.2.6 Dimensional row timestamping

In a much later publication (Kimball (1998a)), Kimball recognises the problem of dimensional counts and appears to have changed his mind about the use of dates. His solution is that the type two method is 'embellished' by the addition of begin and end time stamps to each dimension record. This approach, in temporal database terms, is known as row timestamping. Using this technique, Kimball says it is possible to determine precisely how many customers existed at any point in time. The query that achieves this is as shown in Figure 4.20:

```
Select count(*)
        from Sales_Exec S,Customer C1
        where S. SalesExecNum=C1. SalesExecNum
        And S.Name = 'Tom Sawyer'
        and C.EndDate is NULL
```

Figure 4.20 Counting customers using row timestamping

For the sake of example, the null value in the end date is assumed to represent the latest record but other values could be used, such as the maximum date that the system will accept e.g. 31st December 9999. In effect, this approach is similar to the example in Figure 4.19 because it does not necessarily identify ex-customers and instead of answering the question 'How many customers is Tom Sawyer responsible for?', it may be asking 'How many customers has Tom Sawyer ever been responsible for?. This method will produce the correct result only if the end date is updated when the customer becomes inactive.

The adoption of a row time stamping approach can provide a solution to queries involving counts at a point in time. However, it is important to recognise that the single pair of time stamps is being used in a multi-purpose way to record:

1. Changes in the active state of the dimension.

2. Changes to values in the attributes.

3. Changes in relationships.

Therefore, this approach cannot be used where there is a requirement to implement discontinuous existences where, say, a customer can become inactive for a period because it is not possible to determine when they were inactive. The only way to determine inactivity is to try to identify two temporally consecutive rows where there is a time gap between the ending time stamp of the earlier row and the starting time stamp of the succeeding row. This is not possible to do using standard SQL.

Even where discontinuous existences are not present, the use of row timestamping makes it difficult to express queries involving durations because a single duration, such as the period of

residence at a particular address or the period that they had continuously been an active customer before closing their membership, may be spread over may physical records.

For example, in order to determine how many of the members of the Wine Club had been members for more than a year before leaving during 1998 could be expressed as shown in Figure 4.21:

```
Select '1998' as year, count(*)
    From member m1, member m2
    Where m1.start_date = (select min(m3.start_date)
                from member m3
                where m3.member_code = m1.member_code)
    and m2.end_date = (select max(m4.end_date)
                from member m4
                where m4.member_code = m2.member_code)
    and m2.end_date between '1998/01/01' and '1998/12/31'
    and m1.member_code = m2.member_code
    and m2.end_date - m1.start_date > 365
    group by year
```

Figure 4.21 An example of a query involving a duration

The query in Figure 4.21 contains a self join and two self correlated sub queries. So the same table is used four times in the query. It is likely that the customer dimension is the one that would be subjected to this type of query most often and organisations that have several million customers are, therefore, likely to experience what Kimball calls 'poor browsing performance'.

This problem is exacerbated when used with dimensions that are engaged in hierarchies because, like the type two solution, changes tend to cause the problem of cascaded extraneous rows to be inserted.

The second example has a further requirement which is to determine length of time that a customer had resided at the same address. This requires the duration to be limited by the detection of change events on attributes of the dimension. This is not possible to do with

absolute certainty because of the fact that circumstances, having changed, might later on revert to the previous state. For instance, students leave home to attend university but might well return to the previous address later. In the Wine Club, a supplier might be reinstated as the supplier of a wine they had previously supplied. This is, in effect, another example of a discontinuous existence and cannot be detected using standard SQL.

4.1.3 The 'type three' approach

The third type of change solution (Type Three) involves recording the current value, as well as the original value, of an attribute. This means that an additional column has to be created to contain the extra value. Kimball says that, in this case, it makes sense to add an Effective Date column as well. The current value column is updated each time a change occurs. The original value column does not change. Intermediate values, therefore, are lost. An example as to where this approach is useful is provided by Kimball. Occasionally, there is an abrupt point in time when the names of all the sales districts are changed. For a few months, however, there is a desire to track old history in terms of the new district names and, conversely, to track new history in terms of old district names.

In terms of its support for time, type three is rather quirky and does not add any real value. It will not be considered further in this document. Even Kimball concedes:

> ***'We try to stall on implementing this solution. Usually after a few months people lose interest in the old district names and the requirement goes away'. (Page 104)***

4.2 Review of Bair's approach

Bair (1996) recognises Inmon's definition of a data warehouse but prefers to use the term 'temporal' rather than 'time variant'. In examining the types of DBMS available to support data warehouses, he concludes that the relational model is, in the absence of a specialised temporal solution, the most appropriate DBMS available to provide a solution. Even so he states that relational databases are not really adequate because:

> ***'A data warehouse is a temporal database which depends for support on a relational system which is virtually ignorant of the temporal semantics of the data'.***

Beyond the introduction, Bair does not really concentrate on the temporal issues involved in data warehousing but focuses on general temporal data terms such as valid time, transaction time etc. He provides temporal definitions of tables and types of query

He introduces three classes of temporal data models: Snapshot, Event and State models. Each of the data models can be implemented, in a relational database, using an appropriate type of table. He calls these Snapshot, Event and State tables:

Snapshot table. This is a snapshot as defined in the glossary of temporal terms Jensen et al (1994). It captures the data at a particular instance of time. Snapshot tables can only be regarded as temporal if the data does not change. This means that all snapshots would be equivalent. If the data is expected to change over time, then one snapshot would be different to all others. Snapshot tables have no temporal support at all. Any alterations to the table are applied by logically overwriting the previous values

Event table. According to the glossary, an event is an instantaneous fact. The event model represents temporal data events. Examples of temporal data events are sales, inventory transfers, bookings etc. Event tables are timestamped with the valid-time, if known, of the event. Bair makes the point that event tables are, by nature, 'append only' tables. This is equivalent to the fact table in a dimensional model.

State table. Bair describes a state as 'an instance of temporal data that exists or persists over a period of time'. The existence of states is recorded by the use of a pair of timestamps to mark the start and end of the period for which the state is true. These are intended to delimit the time period during which the state exists. It is recommended that the end-date is given a value of 'forever' where the state is still current. In the physical implementation sense, 'forever' means the highest legal value, for a date, that the DBMS will accept. The pair of timestamps are applied to each row in the table. This is another example of 'row timestamping' as it records the state of an entire row rather than the states of individual attributes. Whereas event tables may be used to represent facts, state tables may be used to represent dimensions.

Bair's approach does not concentrate on dimensional models, as Kimball's does. However, there are some similarities between his and Kimball's logical solution:

Event tables are similar, in both structure and function, to fact tables. He cites an invoice table as an example of an event table. In a dimensional model with a 'sales' subject area, the fact table may well contain invoice details.

He recognises two other tables: snapshot tables and state tables. These are very similar to Kimball's types one and two slowly changing dimensions. Snapshot tables are 'destructive update' tables. This is equivalent to the type one approach. State tables contain rows that are time stamped with a period or a pair of dates that delimit the time for which the state is true. This is similar to Kimball's type two solution. Bair's solution, like Kimball's, does not recognise the requirement for combining attributes that require support for time, together with attributes for which destructive overwrites are appropriate, within the same table.

Where the use of row timestamping is adopted in dimensions, the primary key of the dimension is altered. Either the surrogate key approach has to be adopted, as in Kimball's solution, or part of the timestamp must be included into the primary key to form a composite key. The surrogate key approach has been shown to present problems relating extraneous inserts where the dimension is engaged in a hierarchy. The other approach using row time stamping involves the inclusion of, say, the start date within the primary key. If the primary key is a composite, then the foreign keys that refer to it must also be composite keys. This is not straightforward. The fact table contains a single timestamp that identifies the time that the fact occurred. The join condition from the fact to the dimension would have to be written using the 'between' function to test that the fact date was between the start and end dates of the dimension. Natural joins, therefore, are no longer able to be performed.

Row timestamping provides some advantages but also introduces some difficulties. Whereas Kimball's original type two approach performed well when joining facts to dimensions but did not provide any real temporal support within the dimensional structures, the approach using row timestamping in the dimensions does help with some queries, it makes the fact table to dimension table joins much less precise.

As well as temporal tables, Bair defines three classes of temporal query

1. **Snapshot.** A snapshot query operates on state tables over a single instant of time. The query will select only those rows for which the valid time is true at that instant. A query

such as *'How many Members did the Wine Club have at the end of 1998 ?'* is an example of a snapshot query. Although the query has an implicit temporal aspect, the result set is always non-temporal (a snapshot).

2. **Non-Sequenced temporal.** These queries also operate on state tables. However, there is no implicit temporal component to the query. All the rows in the table, regardless of time, are included. It is difficult to imagine any data warehouse queries that would fall into this category. The concept of state tables implies a time element.
3. **Sequenced temporal.** Sequenced temporal queries operate on state tables over periods of time. The join conditions of any referenced tables include the constraint that their dates must overlap. The result set is a state table where the time is the same as the intersection times of the related tables. An example of a sequenced temporal query is a list of wines that were supplied by a particular supplier at a particular point in time.

Bair also introduces the main temporal selection predicates. These are shown in Figure 4.22 and illustrated graphically in Figure 4.23. In the following section P1 and P2 are both periods with each having a start time and an end time:

P1 < P2	This is true if the start time of P1 is earlier than the start time of P2.
P1 > P2	This is true if the end time of P1 is later than the end time of P2. Note that P1 < P2 and P1>P2 can both be true at the same time.
P1 Precedes P2	The end time of P1 is earlier than the start time of P2.
P1 Meets P2	The end time of P1 is one chronon (Jensen et al (1994)) earlier than the start time of P2, or vice versa.
P1 Overlaps P2	This is true if any part of P1 is the same as any part of P2.
P1 Contains P2	The start of P1 must be earlier than the start of P2 and the end of P1 must be later than the end of P2.

Figure 4.22 Temporal functions

The diagram in Figure 4.23 illustrates these points.

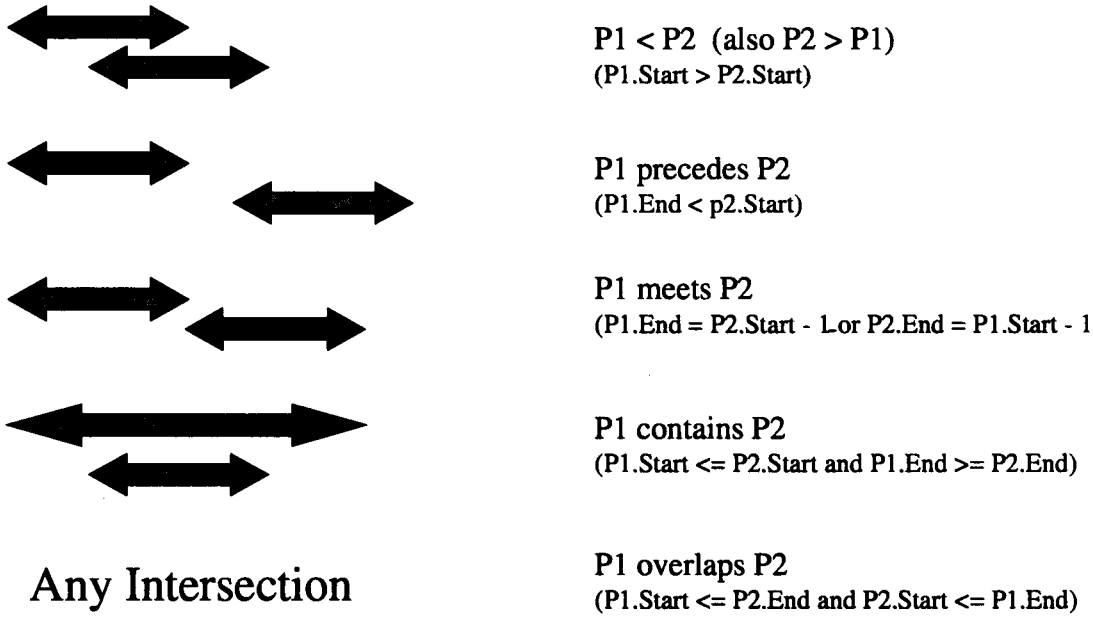


Figure 4.23 Graphical representation of temporal functions

This introduction on how to interpret the TSQL2 functions such as Precedes, Overlaps etc. in standard (non-temporal) SQL is very useful as it shows how these functions can be used in standard SQL.

Upon closer examination, the requirements in a dimensional data warehouse are quite simple. The 'contains' function is useful because it enables queries such as *'How many Members did the Wine Club have at the end of 1998 ?'* to be asked. What this is really asking is whether the start and end dates of a member's period of existence contains the date 31-12-1998. This can be restated to ask whether the date 31-12-1998 is between the start and end dates. So the 'contains' from TSQL2 can be written almost as easily using the 'between' function. Other functions such as 'meets' can be useful when trying to detect dimensional attribute changes in an implementation that uses row timestamps. The technique is to perform a self join on the dimension where the end date of one row meets the start date of another row and to check whether, say, the addresses or supplier codes are different.

4.3 Review of Snodgrass's approach

Snodgrass (1997) recognises that data warehouses are temporal applications. When describing data warehouses and their structure, he has focused on dimensional models and, even more

specifically, on star schemas. In his description of the logical model, Snodgrass says that his approach can be generalised to include snowflake schemas as well as star schemas but that snowflakes are not considered further in his paper.

He describes a star schema as consisting of a single fact table, a time dimension and one or more other dimensions. The dimensions consist of a primary key and other descriptive attributes. Dimension tables are not permitted to contain foreign keys. Snodgrass claims that the fact table is always much larger than the dimension tables and that the implication of this is that a dimension cannot exhibit the same degree of volatility as a fact table. If it did, it would be as large as a fact table and would call into question the reason as to why it was classified as a dimension in the first instance. Thus he asserts that a rapidly changing table must be a *fact* table. There is no quantification of what is meant by rapid nor is there any explanation as to why such a table must be declared as a fact table.

Snodgrass classifies dimensions as either *static*, that is having fixed values over all time, *or slowly changing dimensions*. He uses the term 'slowly changing dimension' to mean the same thing as Kimball although there is a difference in that he does not seem to allow for the type one solution where overwriting previous values is allowed. Like Bair, Snodgrass provides a classification of temporal tables and queries:

Fact tables that contains a foreign key to each of its dimensions. According to Snodgrass:

'the primary key of the fact table is the concatenation of its foreign keys'.

Each row of the fact table denotes something that happened in modelled reality. The event occurred at the time described by the row in the time dimension referred to by the fact. This relates closely to Bair's description of an event table.

Static tables are tables whose attribute values do not change over time. While this might seem unlikely, there are examples in the Wine Club. Such an example would be the region dimension although, in Snodgrass's definition, the region details would be attributes of the wine dimension since logically implemented dimensional hierarchies do not exist in a star schema.

State tables, as with Bair's approach, contain a period attribute. This means that the rows are true for a specified period of time. Another row in the table with the same primary key represents the same dimension at a different time. All the rows with the same key collectively record the history of the dimension. This is, therefore, another example of row timestamping.

Snodgrass's view of state tables is the same as Bair's in that they both adopt the approach of row timestamping so this method will have the same strengths and weaknesses as the other approaches. None of the three methods under review have recognised the requirement for combining attributes that require support for time, together with attributes for which destructive overwrites are appropriate, within the same table.

There are several query types, identified by Snodgrass, that can be executed against a star schema.

1) Dimensional lists

This class of query operates on dimensional values only. It is a dimensional browsing type of query. In the Wine Club, an example would be: *'List the members in the SW region'.*

The SQL for this query is shown in Figure 4.24:

```
Select Member_name, Member_address
    from member
    where sales area_code = 'SW'
```

Figure 4.24 Example of a dimensional list query

If the dimension is implemented as a state table then only the members, when they lived in region 'SW', will be reported. However, as there is no time constraint, the query would be interpreted as 'over all time', so the query would actually translate to: *'List the Members who have, at any time, lived in the SW region even if they no longer members'.* It is not a particularly realistic query in a data warehouse since queries across time do not really make sense. The number of rows returned will depend on how much historical data exists and will continue to grow each time the query is executed.

2) Dimensional lists joined with facts

Here, the same dimension query is executed and the result set is joined to the fact table. The implication is that it is a fact table column that is required to be returned. In the Wine Club, an example of this type of query might be: *'List the sales values of members in the SW region'.*

The SQL is as follows:

```
Select m.member_name, s.value
    from member m, sales s
    where m.member_code = s.member_code
    and sales area_code = 'SW'
```

Figure 4.25 Example of a dimensional and fact list query

Again, as there is no time constraint, the query is really asking for a listing of all sales ever made to members when they lived in the SW region. As with the previous example, it is not realistic because there is no time constraint.

3) Dimensional lists joined with facts with a time constraint

This query is the same as Figure 4.25 but is also constrained by a time group, using the time dimension, as well as a dimensional group. The example might be: *'List the sales values of members in the SW region during 1998'.*

The SQL is shown in Figure 4.26:

```
Select m.member_name, s.value
    from member m, sales s, time t
    where m.member_code = s.member_code
    and t.time_code = s.time_code
    and m.sales area_code = 'SW'
    and t.year = '1998'
```

Figure 4.26 Example of a dimensional list query with a time constraint

The time grouping will only allow sales from 1998 to join to the time dimension. This will automatically constrain the membership records to include only members who lived in SW region during at least part of 1998.

As a data warehousing query, this is more realistic because it is time bounded. It would make still more sense if the results were aggregated as are the next three types of query.

4) Aggregated dimensional lists

This query is similar to query 1 except that the result is an aggregation.

Count the Members who live in the SW region.

The SQL for this query is shown in Figure 4.27:

```
Select Count(*)
    from member
    where sales area_code = 'SW'
```

Figure 4.27 Example of a dimensional counting query

If the dimension table is a state table, the query would be interpreted as 'over all time', so the query would actually translate to:

Count the Members who have, at any time, lived in the SW region.

At first sight, this appears to be a typical data warehousing query. It is a dimensional browsing query which, according to Kimball (1996a), represents eighty percent of all warehouse queries. The missing component is a time constraint. It is unlikely that many queries would be executed, without some kind of time constraint, on a state table. The problem is that there is no way of knowing how much history is being included, so the result set is not really comparable to anything and is of limited value.

5) Aggregated dimensional lists joined with facts

This query is similar to query 2 except that the result is an aggregation '*Sum the sales values of members in the SW region*'.

The SQL is shown in Figure 4.28:

```
Select member_name, sum(Value)
    from member m, sales s
    where m.member_code = s.member_code
    and m.sales area_code = 'SW'
    group by m.member_name
```

Figure 4.28 Example of a dimensional counting query joined with facts

The aggregate is applied to the column from the fact table. Once again, if the dimension is a state table, the query would only really make sense, in a data warehouse application, if a time constraint were added. If the dimension is a state table, then the query would, more realistically, be interpreted as: '*Sum the sales of members that were made when the members were living in SW region over all time*'.

6) Aggregated dimensional lists joined with facts with a time constraint

This query is similar to query 3 except that the result is an aggregation: '*Sum the sales of members that were made when the members were living in SW region in 1998*'.

The SQL is shown in Figure 4.29:

```
Select m.member_name, s.value
    from member m, sales s, time t
    where m.member_code = s.member_code
    and t.time_code = s.time_code
    and m.sales area_code = 'SW'
    and t.year = '1998'
```

Figure 4.29 Example of a dimensional counting query joined with facts and constrained by time

This is very representative of a data warehouse query because it is an aggregation and it includes a constraint on the time dimension. It is now possible to compare the results with others such as different years or different sales areas.

Of the six types of query described by Snodgrass, only two are real data warehouse queries. None of the six are able to answer the question *'How many members do we have ?'* Snodgrass goes on to describe some types of queries that can be executed on state tables only. There are three subclasses of queries involved here:

1) State duration queries – In this type of query, the predicate contains a clause that specifies a period. An example of such a query is: *'List the Members who lived in the SW sales area for a duration of at least one year'.* According to Snodgrass, this type of query selects particular values of the dimension, utilising a predicate associated with a group definition that mentions the duration of the row's period. The problem with his approach here is that a new row is created whenever any of the attributes change. A member might have lived in the SW area for more than a year but may have many records, which have a duration of less than a year, due to changes to attributes other than the sales area. Resolving the query using state tables would mean having to combine many rows to determine the true duration of residence. This is not possible with standard SQL.

2) Temporal Selection Queries – This involves the selection from the state table based on a group definition of the Time dimension. So the following query would fall into this category: *'List the members who lived in SW region in 1998'.* According to Snodgrass, this would involve a join between the state dimension and the time dimension. He says that, with a conventional

star schema, this type of query is not possible since joins between dimensions are not allowed. Kimball (1997b) supports this restriction stating that the time constraints for facts and dimensions are different and, again, uses the 'supplies from stock' problem as a reason. It is possible that Snodgrass has chosen his example carelessly but the solution to this problem could be satisfied, without joins to other dimensions, by the query in Figure 4.30:

```
Select member_name, member_address
    from member
    where (start_date <= 1998-12-31 and end_date >= 1998-01-01)
    and sales area code = 'SW'
    group by member_name, member_address
```

Figure 4.30 Example of a state duration query

This query can be written without having to compare rows. If any one of the rows relating to a member satisfies the predicate, then the member is selected.

3) Transition Detection Queries – In this class of query, we are aiming to detect a change event such as: *'List the members who moved from one region to another region'*. According to Snodgrass, this class of query utilises a 'transition predicate' that identifies consecutive periods for the same dimension. This query is similar to the state duration query in that, in order to write it, it is necessary to compare row values for the same member.

4.4 Variations on a theme

One way of ensuring that the data warehouse correctly joins fact data to dimensions and dimensional hierarchies, is to detach the superordinate dimensions from the hierarchy and re-attach them directly to the fact table.

Pursuing the example presented earlier, in which a Salesman is able to change Departments and a Department is able to move from one Site to another, I present the traditional approach of resolving many-to-many relationships by the use of intersection entities, as the diagram in Figure 4.31 shows:

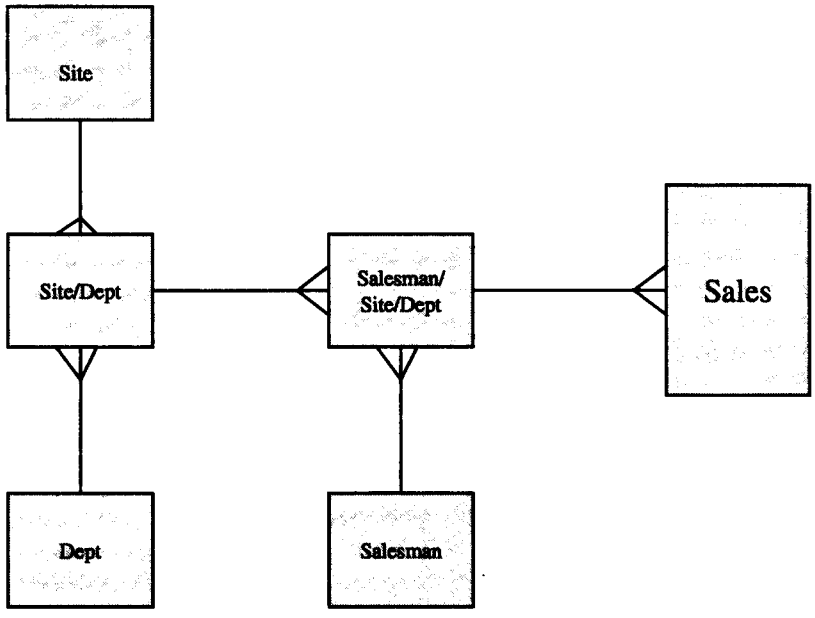


Figure 4.31 Traditional resolution of m:n relationships

This approach need not be limited to the resolution of time related relationships within dimensional hierarchies. It can also be used to resolve the problem of time varying attributes within a dimension. So if it were required, for instance, to track a Salesman's salary over time, a separate dimension could be created as shown in Figure 4.32:

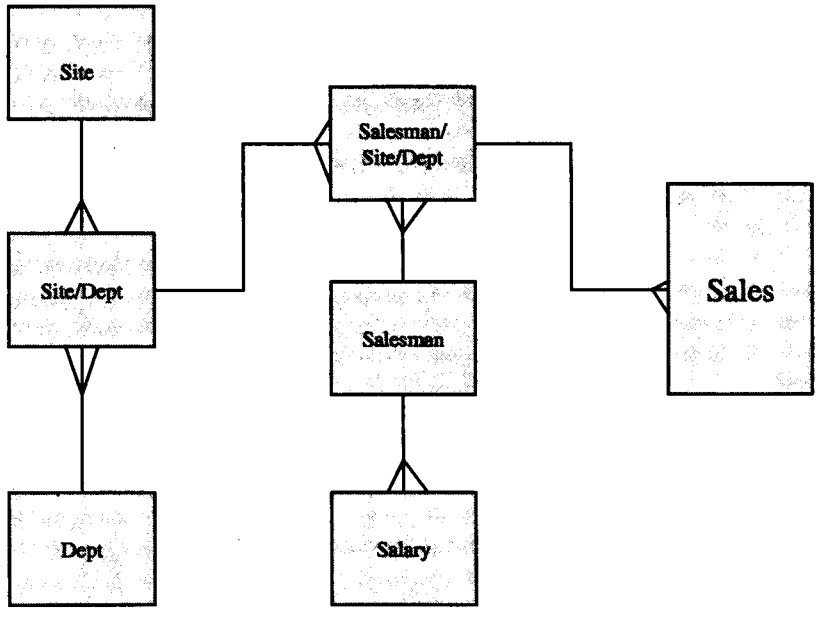


Figure 4.32 Representation of temporal attributes by attaching them to the dimension

The identifier for the Salary dimension would be the Salesman's identifier concatenated with a date. The salary amount would be a non-identifying attribute.

Salary (Salesman_Id, StartDate, EndDate, Salary_Amount)

The same approach could be used with all time-varying attributes.

Another approach is to disconnect the hierarchies and then attach the dimensions to the fact table directly, as is shown in Figure 4.33.

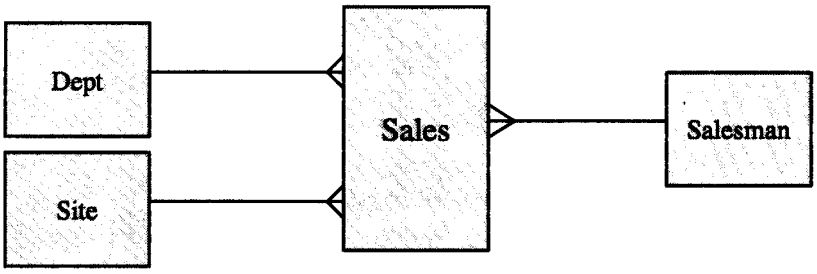


Figure 4.33 Representation of temporal hierarchies by attaching them to the facts

This means that the date that is attached to the fact will, automatically, apply to all the levels of the dimensional hierarchy. The facts would have a foreign key referring to each of the levels of the hierarchy. This approach is recognised by Kamfonas (1998). He refers to it as 'Indirect' versioning. In his paper entitled 'Handling monster dimensions', Kimball (1996f) goes some way toward this solution when he advocates the splitting of large dimensions into two separate dimensions, one of which is relatively static and the second containing the volatile attributes.

We are left with a choice as to how we treat the changing attributes. The salary attribute, in the previous example shown in Figure 4.33, could be treated in the same way as before (i.e. as a separate dimension related to the Salesman dimension). Alternatively, it could be attached directly to the fact table in the same way as the other dimensions as is shown below in Figure 4.34:

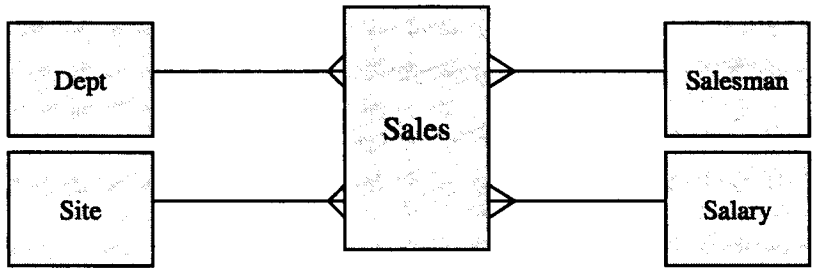


Figure 4.34 Representation of temporal attributes by attaching them to the facts

A further variation on this approach is to include the salary amount as a non-identifying attribute of the fact table, as follows:

Sales(Site_Id, Dept_Id, Salesman_Id, Time_Id, Salary_Amount, Sales_Quantity, Sales_Value)

This approach eliminates dimensional hierarchies and, therefore, removes the problem with type two of extraneous cascaded inserts when changes occur in the hierarchy. However, this is by no means a complete solution as it does nothing to resolve the problems of time within the hierarchical structures. The question *of 'How many customers do we have ?'* is not addressed by this solution. It is presented as a means of implementing type two without incurring the penalties associated with slowly changing hierarchies.

There is a further drawback with this approach. The tables involved in the hierarchy are now related only via the fact table. Therefore the hierarchy cannot be reconstructed so long as no sales records exist for any relationship between the dimensions.

4.5 Conclusion to the review of methods

It has proven very difficult to obtain literature that deals comprehensively with the subject of time in data warehouses. Unlike the conceptual model, where there are no methods for data warehousing, at least in the logical model there are some. Kimball remains the principle proponent. In this chapter, his methods have been subjected to a detailed examination and review. The requirements for the treatment of time in the fact table are accounted for by the inclusion of the time dimension. This is the area in which Kimball's focus has been greatest. The requirements for the treatment of time in the dimensional structures have not, before now, been clearly defined or examined. Kimball recognises that data values within the

dimensions are subject to change but he is focused on ensuring that the dimensional attributes correctly reflect the facts at the time the fact event occurred. In this respect, Kimball appears to adopt the view that the dimensions are really just further attributes of the facts. In this respect, his contribution should be regarded as significant but there are shortcomings in that, where dimensional hierarchies exist, his method does introduce the risk of large numbers of, effectively, duplicated records being inserted into the dimensional tables. Where dimensional hierarchies do not exist, or can be 'modelled out', his method does achieve his objective and does provide an efficient solution. Some of the problems in Kimball's method can be resolved using the variations introduced by Kamfonas (1998). His use of direct and indirect versioning are a useful augmentation of the type two method and their adoption simplifies some queries. However, these variations do not help with the extraneous insertion problem nor with counting and time series questions.

In a much later publication, Kimball has shown that he is beginning to recognise that queries involving dimensions, in isolation from the facts, also need to provide a proper representation of time. His latest approach augments his earlier solution by the introduction of row time stamps. The result is that some straightforward counting queries such as *'How many customers do we have ?'* and time series questions that cannot be answered using Kimball's original method can be addressed using his latest improvements. However, of the three types of temporal query identified by Snodgrass, only 'temporal selection' queries can be addressed in a satisfactory way. The expression of 'state duration' and 'transition detection' queries can be can be achieved in many, but not all, cases. The resulting queries are both very complex to write and very costly to execute. Thus I conclude that the row timestamping solution is only partially suitable for the representation of time in data warehouses.

Bair introduced temporal data models, temporal queries and the temporal extensions in a similar way to TSQL2. His contribution is limited to explaining these terms and it is left to the reader to interpret their usefulness in a data warehousing context. His translation of the temporal functions into standard SQL is very helpful and much use will be made of this in chapter five. Bair also supports the idea of row timestamping as a method of tracking time in a dimension table.

Snodgrass has adopted a similar solution to Bair in that he recommends a row timestamping approach. He does at least attempt to apply his approach to dimensional data warehouse but specifically avoids the complications involved in dimensional hierarchies and focuses entirely on star schemas. From his list of six query types, only two were typical of data warehousing. He also tries to classify some dimensional queries but, due to the implementation using row timestamping, he is forced to use a temporal query language to write comprehensible queries. Snodgrass's classification of temporal queries into temporal selection, state duration and transition detection will be carried forward and used to define temporal queries from now on.

In conclusion, the original type two solution does not provide a method for the proper representation of time in data warehousing. The use of row timestamps does provide a partial solution but the queries are difficult to write and are likely to be costly to execute. It also suffers from the problem of cascaded insertions and does not lend itself to the combined requirements where some components of the data warehouse need support for time while others do not and where the type one approach is appropriate.

The problems associated with the representation of time in data warehouses that were described in chapter two have been only partially addressed by the solutions under review. A solution that satisfies all the requirements relating to accuracy, and is relatively easy to use, is still needed.

5 A model for time in data warehouses

5.1 Introduction

So far, as a result of the research undertaken and described in this thesis, the problems associated with the representation of time in data warehouses have been identified and the solutions that are used currently have been investigated and reviewed. The temporal ER models did not enable all the requirements to be recorded and, in any case, they were found to be too complex to be used by inexperienced or non-technical people.

In this chapter of the thesis, I will address the problems described in chapter two and improve upon the solutions that were explored and reviewed in chapters three and four. The objective is to:

- Provide an underlying theory that explains how the problems can be solved.
- Assess the impact of the theory on current practice.
- Show how the theory might be implemented in practice.

Existing solutions, or modified versions thereof, that have been reviewed in the preceding chapters will be used where I can show that they add value.

The most important property of the solution is that it enables almost any required level of accuracy to be achieved, depending on the amount of effort and computer resource applied. A further property of the solution is that it allows the complex requirements for time to be recorded in a straightforward manner that does not assume a great deal of knowledge on the part of the user.

This chapter is divided into two sections. The first concentrates on the general issues and classification of time as it affects data warehousing. The second section focuses on the conceptual modelling part of the solution. In this respect we adhere to the definition of the conceptual model as one that specifies the information requirements. This is based on providing the following components:

- A diagram depicting the major data objects.

- The temporal requirements of each data object, including attributes, using the classification already defined.

- The interdependence of data objects and the causal nature of changes.

Model Component Naming Conventions

In view of the specific nature of dimensional models, together with the behavioural aspects of some of the components, it is desirable to redefine the components so that it is clear which aspects of the model are being discussed.

The central entity that contains the facts to be measured will be referred to as the *Facts*. In the Wine Club, the Facts are Sales.

The entities that are *directly* attached to the Facts will be referred to as *Dimensions*. The relationships between the Facts and Dimensions have a cardinality of one-to-many from the Dimension to the Facts. In the Wine Club example Wine, Member and Time are the only Dimensions.

Dimensions are sometimes grouped into hierarchies. These will be referred to as *Dimensional Hierarchies*. Dimensional hierarchies may be single level or multi-level. In the Wine Club example Supplier → Wine and Hobby → Member are single level dimensional hierarchies and Manager → Sales Area → Member is a multi-level dimensional hierarchy. The individual components of a dimensional hierarchy will be referred to as *Groupings* of dimensions. Supplier, Region and Sales Area are *First Level Groupings*. Manager is *a Second Level Grouping*. Higher level groupings are possible but none are shown in the Wine Club example.

Dimensions, dimensional hierarchies and groupings can be referred to collectively as dimensions.

Relationships will continue to be referred to as such. However, relationships:

1. Will not be named

2. Have implicit cardinality of one-to-many from the outside inwards towards the facts. The dimension farthest from the facts is the 'one' end of the relationship and the dimension nearest to the facts is the 'many' end.

3. Have implicit participation of optional at the 'one' end and mandatory at the 'many' end.

Attributes will continue to be referred to as such.

5.2 The classification of temporal requirements in data warehousing

5.2.1 Introduction

I have established that the creation of a time variant system that has, as its source, data from time invariant systems causes disruption in the relationships between entities and other entities as well as relationships between entities and attributes. The link between 'time variant' and 'temporal' has not been widely acknowledged by practitioners. From experience, I believe that there is a general lack of awareness rather than a lack of acceptance or agreement of temporal issues outside of the academic field. With the exception of Bair (1996) and Snodgrass (1997), none of the literature makes the connection between data warehousing and temporal databases. In order to enable practitioners to engage in data warehouse development projects with a full understanding of the temporal issues involved, the temporal classifications, as they relate to data warehousing are now made clear. I start by revisiting the definition of the principal property of a data warehouse with respect to time.

Within the industry, some confusion has arisen around the terminology. Two sets of terms have emerged. These are:

1. Time Variant vs. Time Invariant

2. Consistent History vs. Inconsistent History

These two sets of terms are supposed to be synonymous. One of the terms in each set describes the situation in which history is reflected faithfully and the other describes the situation in which history is lost. The problem is that different practitioners have interpreted both pairs of terms differently. It is not the intention of this document to say which is right or

wrong. In any case, after examination, I have discovered that there are three options, not just two.

There is a general confusion that emerges as the problem is researched. Most of the literature describes two alternatives with respect to the treatment of time. The first is the case where it is required to faithfully record the history of existence of an entity, a relationship or the value of an attribute so that queries return a true view of the historical values.

The second case varies from one approach to the other. Some, such as Snodgrass (1997), interpret the second case to be that the values are permanent and never change. Others, such as Bair (1996) on the other hand, take the view that the second case is where the new values simply replace the old values so that the old values are lost. However, when we examine the requirements for data warehousing, it is clear that either of these cases can occur and, in most implementations, both will occur.

Insofar as dimensional models are concerned it is very useful to identify three alternatives as it will help with the logical and physical design if we know which of the three types applies to each dimension, relationship and attribute in the system. This is dealt with in the section covering the treatment of dimensions.

5.2.2 The treatment of facts

As has been previously explained, the facts in a dimensional model are the central focal point in the model. The data warehouse subject area often has the same name as the facts. The facts may be simple atomic level transactions or they may be aggregations of more granular facts.

The temporal requirements of facts are very straightforward. Once recorded, facts do not change. They do not, therefore, have any form of lifespan. Facts are usually derived from an entity that has attained a particular state. The attainment of the state is caused by an event and the event will occur at a particular point in time. So, one of the attributes of the fact will be a time attribute. The time attribute records the time of the occurrence of the event. The time attribute will be used in two ways:

1. As part of the selection constraint in a query.

2. As an aid to joining the dimensions to higher level groupings.

For each event or change there is, as has been said, an associated time. Different applications have different requirements with respect to the grain of time. Some applications require only a fairly coarse grain such as day. These might include:

- Insurance companies selling motor insurance
- Banks analysing balances
- Supermarkets analysing product-line sales

Some other applications might wish to go to a finer grain, perhaps hours. These might include:

- Supermarkets analysing customer behaviour
- Transport organisations monitoring traffic volumes
- Meteorological centres studying weather patterns

Other applications would require a still more fine grain of time to say, seconds. An example of this is the telecommunication industry monitoring telephone calls. The data model needs to show clearly the granularity of time pertaining to the model.

Some applications require more than a single grain of time. Consider the examples above. The supermarket example shows a different requirement with respect to time for product analysis as for customer behaviour analysis. In any case, almost all organisations conducting dimensional analysis will require the ability to summarise information, from whatever the granularity of the base event, to a higher (coarser) level in order to extract trend information.

It has been observed that many of the emerging temporal modelling techniques contain the disadvantage that the essential data and relationships can become obscured when time is introduced into the model. Dimensional models are relatively quite simple and the simplicity

must not be lost when time is introduced because, as Codd et al (1993) confirm, the ability of non-technical people to understand the model is an important requirement.

5.2.3 The treatment of dimensions - retrospection

In a dimensional data warehouse that uses Kimball's type two method, the way in which an attribute of a dimension is treated, with respect to historical values, depends entirely upon the requirements to partition the facts in historical terms. According to Kimball (1996a), the key role of dimensional attributes is: *To serve as the source of constraints in a query.* The requirements for historical partitioning of dimensional attributes for, say, dimension browsing have been regarded as secondary considerations. Principally, the dimensional attributes exist to constrain queries about the facts. According to Kimball, even though some eighty percent of queries executed are dimension browsing, the main business purpose for this is as a process of refinement of the fact table constraints. It is only comparatively recently (Kimball 1998a) that he has begun to accept that the dimensions themselves yield valuable information about the business such as the growth in customers etc. In some industries (especially telecommunications), this is seen as the largest business imperative at the present time.

In recognition of the need to place more emphasis on the treatment of history within the dimensional structure, we have to examine the model in detail in order to assess how each of the various elements should be classified.

Each dimension, relationship and dimensional attribute from the model, which is subject to change, will be evaluated to assess the way in which past (historical) values should be treated. Each will then be given a classification. This is called the ***'Retrospection'***[4] of the object.

It follows from section 5.2.1 that retrospection has three possible values:

1. True

2. False

3. Permanent

[4] Retrospection means literally – 'looking back into the past'.

True retrospection means that the object will reflect the past faithfully. It enables queries to return temporal sub-sets of the data reflecting the partitioning of historical values. Each dimension, relationship and attribute value will, in effect, have a lifespan that describes the existence of the object. An object may have a discontinuous lifespan. That is: many periods of activity, punctuated by periods of inactivity. True retrospection is the most accurate portrayal of the life of a data warehouse object.

False retrospection means that the view of history will be altered when the object's value changes. In simple terms, when changes occur, the old values will be overwritten and are, therefore, lost. It is as though the old values had never existed.

Permanent retrospection means that the value of the object will not change over time.

I will now describe how the various values for retrospection apply to dimensions, relationships and attributes.

5.2.3.1 Retrospection in dimensions

So far as dimensions are concerned, the value for retrospection relates to the *existence* of the dimension. The existence of, for instance, a wine starts when the wine first becomes available from the Wine Club and the existence ends when the wine is no longer available.

Retrospection = *true* for dimensions means that the lifespan of the dimension consists of one or more time intervals. A single wine may not have a single, continuous lifespan. It may be available for intervals of time spanning many years or the entire lifespan may be punctuated by periods when the wine is not available. An example of this would be the Beaujolais Nouveau which, for some reason, is very popular when first available in November each year but must be consumed quickly as it very soon deteriorates further. As this wine is not available for ten months out of each year, it is reasonable to say that the lifespan of this wine is discontinuous.

Retrospection = *false* for dimensions means that the current state only, of the existence of the dimension, is recorded. An example from the Wine Club would be the Supplier dimension. There may be a need to be able to distinguish between current suppliers and previous suppliers. There is no requirement to record the intervals of time when a supplier was actually supplying wine to the Wine Club as distinct from the intervals of time when they were not.

Retrospection = *permanent* for dimensions means that the dimension exists forever. The concept of existence does not, therefore, apply. An example from the Wine Club would be the Region dimension. Regions, which represent the wine growing areas of the world, are unlikely to disappear once created.

5.2.3.2 Retrospection in relationships

In a dimensional model, the degree of snapshot relationships is always 'One-to-Many'. When the relationship becomes temporal, due to the need for true retrospection on the relationship, the degree of the relationship may change to that of 'Many-to-Many'. This has been described in detail in chapter two of this thesis.

It is important that this information is recorded in the model without, as has been previously stated, introducing significant complexity into the model. The essential simplicity of the model must not be dominated by time, while at the same time it needs to be straightforward for designers to determine, quickly and precisely, the degree to which support for temporal relationships is required.

The situation with relationships is similar to that of dimensions. It is the requirement with respect to the existence, and lifespan, of a relationship that determines the value for retrospection in relationships.

Retrospection = *true* for relationships means that the lifespan of each relationship must be recorded and kept so that the results from queries will faithfully reflect history. An example of this in the Wine Club is the relationship between Wine and Supplier. As the club switches from one supplier to another for a particular wine, it is important that the previous relationships of that wine with other suppliers are retained.

Retrospection = *false* for relationships means that only the current relationship needs to be recorded. There is no need for the system to record previous relationships. A true view of history is not required. An example of this, within the Wine Club, is the relationship between a hobby and a member. If a member informs the club, through the periodic data update process, that their hobby has changed from, say, horse riding to choral singing, then the new hobby replaces the old hobby and all record of the old hobby for the member is lost.

Retrospection = *permanent* for relationships means that the relationship is never expected to change. No change procedures have to be considered. An example of this kind of relationship, in the Wine Club, is the relationship between a Wine and a growing region. A wine is produced in a region. A particular wine will always be produced in the same region so it is reasonable to take the view that this relationship will not change.

5.2.3.3 Retrospection in attributes

Each attribute in the model must be assessed to establish whether or not it needs temporal support. Therefore, one of the properties of an attribute is its value for retrospection. As with the other data objects, the recording of the temporal requirements for attributes should not introduce significant complexity into the model.

The situation with respect to attributes and retrospection appears, at first, to be somewhat different to the requirements for other objects. In reality the situation is very similar to that of relationships. If we consider an attribute to be engaged in a relationship with a set of values from a domain, it becomes easy to use precisely the same approach with attributes as with relationships.

Retrospection = *true* for attributes means that we need to record faithfully the values associated with the attribute over time. An example of this is the cost of a bottle of wine. As this cost changes over time, we need to record the new cost price without losing any of the previous cost prices.

Retrospection = *false* for attributes means that only the latest value for the attribute should be kept. When the value of the attribute changes, the new value replaces the old value such that the old value is lost permanently. An example of this in the Wine Club is the Alcohol By Volume (ABV) value for the wine. If the ABV changes, then the previous ABV is replaced by the new ABV.

Retrospection = *permanent* for attributes means that the value is not expected to change at all. Changes to the values of these type of attributes do not have to be considered. An example of this type of attribute in the Wine Club is the Hobby Name. Once a hobby has been given a name, it is never expected to change.

There is a rule that can be applied to identifying attributes. With great originality it is called the 'Identifying Attribute Rule' and simply states that all identifying attributes have a value for retrospection of '*permanent*'. This is because the identifying attributes should not change.

There is an implicit inclusion of an existence attribute for dimensions, relationships and attributes where the value for retrospection is true. The status of true retrospection will direct the logical database designers to provide some special treatment, with respect to time, to the affected object. The type of treatment will vary on a per-case basis and will, to some extent, depend on the type of database management system that is to be deployed.

The inclusion of an existence attribute is also implicit for dimensions, but not relationships and attributes, where the value for retrospection is false. The provision of time support, where retrospection is false, is usually simpler to implement than where retrospection is true. For relationships and attributes, it is simply a case of replacing the previous value with a new value. In other words, a simple update. The treatment of relationships can be very similar to that of attributes, as is the case in relational implementations, or it may be very different in some of the proprietary multi-dimensional database management systems.

5.2.3.4 Retrospection and the case study

Figure 5.1 now lists the data objects, facts, dimensions, dimensional attributes and relationships for the Wine Club. For each, the value for retrospection is given that satisfies the requirements of the Wine Club with regard to the representation of time.

In accordance with the previous point about their implicit nature, there is no explicit mention of existence attributes in the following list.

Object Name	Type	Retro-spection	Reason
Colour	Dim	Permanent	The colours are assumed to exist forever
Time	Dim	Permanent	The days, once entered, will exist forever. Each year, a new 'years-worth' of dates will be added but their existence, thereafter, is permanent.
Hobby	Dim	Permanent	The hobby details, once entered,

			will exist forever.
Manager	Dim	True	The Wine Club wishes to monitor the performance of managers over time. So the history of a Manager's existence is needed.
Member	Dim	True	Members may have more than one interval of membership. It is important to the Wine Club that it monitors the behaviour of members over time. There is a requirement, therefore, to record the full details of the existence of members.
Month	Dim	Permanent	The months, once entered, will exist forever. Each year, a new 'years-worth' of dates will be added but their existence, thereafter, is permanent.
Region	Dim	Permanent	Wine growing regions are not expected to cease to exist
Sales	Fact	Permanent	Fact table entries exist permanently
Sales_Area	Dim	False	Latest existence only, is required. Sales areas may be combined, or split. Only the latest structure is of interest.
Supplier	Dim	False	Latest existence only, is required. The supplier details are required, but no history.
Wine	Dim	True	Wines may have an discontinuous existence as far as the Wine Club is concerned. It is important to track the history of the existence of wine. Questions such as 'How many wines do we sell today, compared to a year ago ?' can be answered accurately only if we keep track of each wine's existence.
Colour→Wine	Rel	Permanent	The colour of a wine will not change over time. This is a permanent property of the wine.
Region→Wine	Rel	Permanent	The growing region of a wine will not change over time. This is another permanent property of the wine.

Supplier→Wine	Rel	True	The suppliers of wines will vary over time. One of the objectives of the Wine Club is to monitor the performance of Suppliers with respect to the popularity and quality of the wines they supply. Where necessary, the club will switch suppliers for wines. So there is a need to monitor this relationship over time.
Wine→Sales	Rel	Permanent	The relationship of a particular sale to the wine involved in the sale will never change.
Manager→Sales Area	Rel	True	Managers do move from sales area to sales area. There is a requirement to monitor the performance of managers. Therefore it is important to keep track of history of their involvement with sales areas.
Sales Area→Member	Rel	True	There is a requirement to monitor the performance of sales areas. As members move from one area to another, therefore, we need to retain the historical record of where they lived previously, so that sales made to those members can be attributed to the area in which they lived at the time.
Hobby→Member	Rel	False	A member's hobby is of interest to the Wine Club. Only the current hobby is required to be kept.
Member→Sales	Rel	Permanent	The relationship of a particular sale to the member involved in the sale will never change.
Time→Sales	Rel	Permanent	The relationship of a particular sale to the date involved in the sale will never change.
Colour.Colour_Code	Att	Permanent	Identifying Attribute Rule
Colour.Colour	Att	Permanent	The colour never changes
Time.Time_Code	Att	Permanent	Identifying Attribute Rule
Time.Day_Name	Att	Permanent	The value will never change
Time.Week_End	Att	Permanent	The value will never change
Time.Week	Att	Permanent	The value will never change
Time.Month	Att	Permanent	The value will never change
Time.Month_Name	Att	Permanent	The value will never change

Time.Season	Att	Permanent	The value will never change
Time.Year	Att	Permanent	The value will never change
Sales_Area.Sales_Area _Code	Att	Permanent	Identifying Attribute Rule
Sales_Area.Sales_Area _Name	Att	False	The latest value only, is sufficient
Manager.Manager_Code	Att	Permanent	Identifying Attribute Rule
Manager.Manager_Name	Att	False	The latest value only, is sufficient
Hobby.Hobby_Code	Att	Permanent	Identifying Attribute Rule
Hobby.Hobby_Name	Att	Permanent	The value will never change
Member.Member_Code	Att	Permanent	Identifying Attribute Rule
Member.Member_Name	Att	False	The latest value only, is sufficient
Member.Member_Address	Att	True	Requirement to analyse by detailed area down to town/city level
Member.Date_Joined	Att	False	The latest value only, is sufficient
Region.Region_Code	Att	Permanent	Identifying Attribute Rule
Region.Region_Name	Att	Permanent	The value will never change
Region.Country	Att	Permanent	The value will never change
Supplier.Supplier_Code	Att	Permanent	Identifying Attribute Rule
Supplier.Supplier_Name	Att	False	The latest value only, is sufficient
Supplier.Supplier_Address	Att	False	The latest value only, is sufficient
Supplier.Supplier_Phone	Att	False	The latest value only, is sufficient
Wine.Wine_Code	Att	Permanent	Identifying Attribute Rule
Wine.Wine_Name	Att	False	The latest value only, is sufficient
Wine.Vintage	Att	True	There is a requirement to analyse popularity of wine by vintage. The vintage of a wine changes from time to time, approximately yearly.
Wine.ABV	Att	False	The latest value only, is sufficient
Wine.Bottle_Price	Att	True	There is a requirement to analyse popularity by price ranges and to determine how changes in price affect popularity
Wine.Case_Price	Att	True	There is a requirement to analyse popularity by price ranges and to determine how changes in price affect popularity
Wine.Bottle_Cost	Att	True	Requirement to analyse changes in cost vs. revenue

Figure 5.1 Wine Club dimensions, groupings and attributes

In summary, I have analysed the requirements with respect to time and have identified three cases that can occur. The data model has been examined and each object classified accordingly.

The requirement, which follows on from this, is to develop a theoretical method that enables the classification to be incorporated into the model so that a solution can be designed. It is important that the requirements, at this level, do not prescribe a solution. The designers will have additional points to consider such as the volumes of data, frequency of access and overall performance. As always, some compromises are likely to have to be made.

5.2.5 The identification of changes to data

5.2.5.1 Causality

It would be very helpful if, during the analysis of the kinds of changes that can occur, it is made clear as to whether the changes are causal or not causal in nature. It would be sufficient to identify causal changes only and to assume that all unidentified changes are non-causal. This will provide assistance to the designers. Some changes, as has been previously indicated, can occur to attributes that have the property of false retrospection but which, due to the fact that they are determinants, have a 'knock-on' affect on other attributes that might have the property of true retrospection. The capture of changes has to be developed into an automated process. Some mechanism is created that enables changes that have occurred to be identified by examining the organisation's operational systems as these are, invariably, the source of data for the data warehouse. The source system will not share the data warehouse data model and will not be aware of the affect of changes. For instance, in the Wine Club, there is a relationship between the address of a member and the sales area that contains the address. So it could be said that the address determines the sales area. This determinant relationship is identical to that used in the process of normalisation. However, the purpose here is quite different and has more to do with the synchronisation of the timing of changes to attribute values, to ensure temporal consistency, than the normalisation of relations. Thus the term 'causality' has been adopted in order to distinguish this requirement as it is unique to data warehousing. The operational system that records members' details may not be aware of the sales area hierarchy. When a member moves address, the fact that a change in sales area might have occurred would not normally be apparent. It becomes the responsibility of the data warehouse designer to manage this problem.

There is very little information on this subject in the literature. Brackett (1996) recognises that logically connected data may need to be extracted from different data files which, in turn, might belong to various operational systems. That there may be a need to implement a

physical link between data sources due to the causal nature of the relationship is not recognised. Poe et al (1997) calls this linkage of disparate sources 'data consolidation' and describes it as a data modelling activity. Kimball (1996a) describes the problem of what he calls 'incompatible granularity' under the heading of 'conforming dimensions' when trying to combine data from disparate sources. Kelly (1996) introduces a term called 'data enrichment' where new attributes are added to existing data due to the introduction of, in his example, external data. In this case, an external attribute relating to a customer's economic classification is added to the customer's record. This is a good example of causality. What is the trigger that generates a change in the economic classification when a change in the customer's circumstances is implemented so that temporal consistency is maintained?

Without such a facility the data warehouse may be recording inconsistent information. If a member changes their address, then the sales area code must be checked and updated, if necessary, at the same time as the address change. Where the data relating to addresses and sales areas is derived from different source systems, the temporal synchronisation of these changes may be difficult to implement. If temporal synchronisation is not achieved, then any subsequent query involving the history of these attributes may produce inaccurate results.

None of the authors reviewed have covered this issue. The main point is to recognise the problem and ensure that the causal nature of changes is covered during the requirements analysis.

5.2.5.2 The frequency of capture of changes

Associated with identification of changes is the timing with which changes to data are captured into the data warehouse. In this respect, the behaviour of facts is different to the behaviour of dimensions. The frequency of capture for the fact data is usually as close as possible to the *granularity* of the valid time event. For instance, in the Wine Club example, the granularity of time of a sale is 'day' and the sales are captured into the data warehouse on a daily basis. There are exceptions, such as telecommunications where the granularity of time is 'seconds' but the frequency of capture is, typically, still daily. Nevertheless, the time assigned to the fact can usually be regarded as the valid time.

The granularity of time for recording changes to the dimensions adopts an appearance that is often misleading. The most frequently used method for identifying changes to dimensions is by use of the file comparison approach, as outlined in chapter 2.5.2 of this thesis. The only time that can be used to determine when the change occurred will be the time that the change was detected (i.e. the time the file comparison process was executed). The time recorded on the dimensional record can be at any level of grain e.g. day. In this example, the granularity of time for the changed data capture appears to be daily because the date that the change was captured will be used to record the change.

However, this is a record of the transaction time so far as the data warehouse is concerned. It is not actually a record of the transaction time that the change was recorded in the originating source system. The granularity is related to the frequency that the changed data is captured. If the changes are detected and captured into the data warehouse on a monthly basis, then the transaction time frequency should be recorded as monthly.

In practical situations, different parts of the dimensional model are usually updated at differing frequencies of time. Some change data is captured daily while others are weekly and, still others, monthly. The frequency of capture is often dependent on the processing cycle of the source systems.

As with the previous section on causality, the valid time and transaction time should be the same, if possible. Where such synchronisation is not possible the difference between the two times should be recorded so that the potential error can be estimated. None of the modelling methods provided a means of capturing the true granularity of time on a 'per attribute' basis.

5.3 Conceptual Model

5.3.1 Requirements of the conceptual model

Before proceeding, the general requirements of a conceptual data model for data warehousing are re-stated succinctly. The model must provide for the following three requirements:

1. Support for time

2. Support for dimensional schemes

3. Be simple to understand and use

There is also a practical objective of this thesis and that is to produce a practitioner's guide to developing data warehouses that fully incorporates the representation of time and that takes advantage of the underlying theory. The conceptual model will be an important component of this guide.

In the previous chapter, the review of candidate conceptual models concluded that most of them were quite complex. In general, they were enhanced versions of the entity relationship (ER) or extended entity relationship (EER) methods. The enhancements required to provide temporal support tended to add to the complexity by the introduction of additional notation. In many respects, however, the diagrammatic requirements for data warehouses are simpler than traditional ER models:

1. The structure of a dimensional model is predictable. There is a single fact table at the centre of the model. The fact table has one or more dimension tables related to it. Each dimension will have zero, one or more hierarchical tables related to it.

2. The relationships are not usually complex. Relationships are always 'one-to-many' in a configuration where the dimension is at the 'One' end of the relationship and the fact table is the 'Many' end. Where dimensional hierarchies exist, the outer entity (farthest from the fact table) is the 'One' end and the inner entity (nearest to the fact table) is the 'Many' end.

3. 'One-to-One' and 'Many-to-Many' relationships are rare, although this changes when 'time' is introduced. There is no real need to model the cardinality (degree) of the relationships.

4. The participation conditions do not need to be specified. The dimension at the 'One' end of the relationship always has optional participation. The participation condition for the dimension, or fact, at the 'Many' end is always mandatory.

5. Entity Super/Sub Types do not feature in dimensional models

6. There is no requirement to name or describe the relationships as their meaning is implicit. It is important to show how the dimensional hierarchies are structured but that is the only information that is needed to describe relationships.

7. There is no requirement for the fact table rows to have a unique identifier.

8. There is no requirement to model inclusive nor exclusive relationships.

The additional rules and notations that are required to support the features in the list above are, therefore, not appropriate for dimensional data warehouses.

There is a further consideration. I have explained that data warehouses are not designed to support operational applications such as order processing or stock control. They are designed to assist business people in decision making. It is important, therefore, that the data warehouse contains the right information. Often, the business people are unable to express clearly their requirements in information terms. It is frequently the case that they feel they have a problem but are unsure where the problem lies. This issue was brought out in the introduction to the Wine Club case study in Chapter 1, Section 3.2. This uncertain type of problem is similar to that described by Checkland (1981) and Checkland and Scholes (1990).

'We think we've got problems but are unsure what they are' (Checkland 1981 p.154)

Most business managers have a set of business objectives. These can be formally defined 'key performance indicators' (KPIs), against which their performance is measured, or they can be more informal, self imposed, objectives. A data warehouse can be designed to help them to

achieve their business objectives if they are able to express them clearly and to describe the kind of information they need to help make better decisions in pursuit of their objectives. One method that leads business managers through a process of defining objectives and subsequent information requirements is described in the practitioners' guide in Appendix A of this thesis.

Checkland's approach has been to develop the 'Soft Systems Methodology' (SSM) as an approach to model these 'social' problems. The main component of an SSM model is a 'Rich' picture (Checkland and Scholes P.46) that models the major processes involved in the system. In this respect the approach is similar to the structured techniques used in software development such as Yourdon's data flow approach (see DeMarco 1979) and attempts have been made to integrate SSM with the UK Governments 'Structured Systems Analysis and Development Method' (SSADM – see Checkland and Scholes P.58). This 'process' oriented approach is not suited to data warehouse requirements because we need to describe the system in terms of its state; that is the elements that comprise it and their relationships. A data warehouse is a repository of data organised in a particular way. It cannot be described as a set of processes. Checkland (1981, P.169) recognises this distinction but does not pursue this latter requirement.

What is needed is an abstraction that allows for the business requirements to be focused upon in a participative fashion. The business people must be able to build, validate, modify or even replace the model themselves. However, in addtion, the model must be powerful enough to enable the temporal requirements of each data object to be specified so that the data warehouse designers can go on to develop the logical model.

The next section introduces the Dot modelling notation as a model for capturing information requirements in a way that business people can understand. There exists a fundamental requirement that the people who use the data warehouse must understand how it is structured. This is what Codd meant when he said the 'dimensional conceptual view' represents the way business people view their organisation. It is also what Kimball (1998a) means when he says that dimensional models:

'Always appear when the designer places understandability and performance as the highest goals'. (Page 143)

Some business applications need to be supported by complex data models. Data modelling specialists are comfortable with this complexity. Their role in the organisation, to some extent, depends on their being able to understand and navigate complex models. The users of the systems, usually parametric in nature, are shielded from the underlying complexity by the human computer interface. The users of data warehouses are usually business people such as marketing consultants. Their usage of the warehouse is often general and, therefore, unpredictable and it is not necessary or desirable to shield them from the structure of it. The conceptual model should be easy to understand by non-technical people to the extent that, with very little training, such people could produce their own models. If it is accepted that the dimensional model, due to its simplicity, is an appropriate method to describe the information held in a data warehouse, then it would be sensible to ensure that the simplicity is maintained and that the model does not add complexity. As Checkland and Scholes say:

'To be usable on the hoof throughout a study, a model has to be very simple indeed' (p.49)

Another requirement of the conceptual model is that it should retain its dimensional shape. Again, having achieved a model that is readily understood, we should try to ensure that the essential radial shape is retained even in relatively complex examples.

With respect to the notation, it would be possible to include some form of symbolic representation on the diagram, such as the '⏲' symbol, used by Zimanyi (1997), or the 'T' symbol, preferred by McBrien (1992), and this approach was considered very carefully. However, for the sake of simplicity, it is preferable that the diagram should contain only facts and dimensions. However, time also affects the attributes in the model. In order for the diagram to be complete insofar as the representation of time is concerned, the attributes would need to be included on the diagram. This issue was considered in chapter three and the conclusion drawn was that the methodology should be consistent. That is, it should be capable of being used in all cases where a dimensional data warehouse is being developed. So it is not really sensible to take one approach for one class of model where, say, the number and size of dimensions makes it possible to include all the information on a single diagram without resulting in the diagram becoming unduly complex while a second model, perhaps with larger and more numerous dimensions, cannot reasonably be accommodated in the same way.

Also, there is a need within the model to record business semantic information about each of the attributes. This means that some additional, supporting, information about attributes will have to be recorded in any case. It would seem sensible, therefore, to include information about time within the supporting documents. In consideration of the conclusions drawn and, having considered this issue with respect to the solution, I have decided that the representation of time should be omitted from the diagram and dealt with in the supporting documentation.

5.3.2 Dot Modelling

5.3.2.1 Introduction

Dot modelling is proposed in order to provide a complete methodology for the design of data warehouses. It is based on the simplified requirements for dimensional models that were described in the introduction to this chapter. It is a complete methodology that enables non-technical people to build their own conceptual model that reflects their personal perception of their organisation in dimensional terms. It also provides a structured way for constructing a logical (currently relational) model from the conceptual. The conceptual → logical mapping takes account of all the temporal issues.

The method was invented in July 1997 and has been used in real projects since then. It has received positive reviews from non-technical people in environments where it has been deployed. The name was given by a user. Dot is not an acronym. It comes from the characteristic that the centre of the model 'the facts' are represented by a dot. The method was developed as a kind of evolution using dimensional concepts as follows. The diagram in Figure 5.2 represents the design of a two dimensional tabular report. This kind of report is familiar to everyone and is a common way of displaying information e.g. as a spreadsheet.

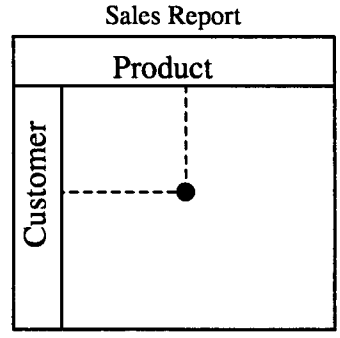


Figure 5.2 Example of a two dimensional report

The intersection of the axes in this example, as shown by the dot, would yield some information about the sale of a particular product to a particular customer. The information represented by the dot is usually numeric. It could be an atomic value, such as the monetary value of the sale, or it could be complex and may include other values such as the unit quantity and profit on the sale.

Where there is a requirement to include a further dimension such as time into the report then one might envisage the report being designed as several pages where each page represents a different time period. This could be displayed as shown in Figure 5.3:

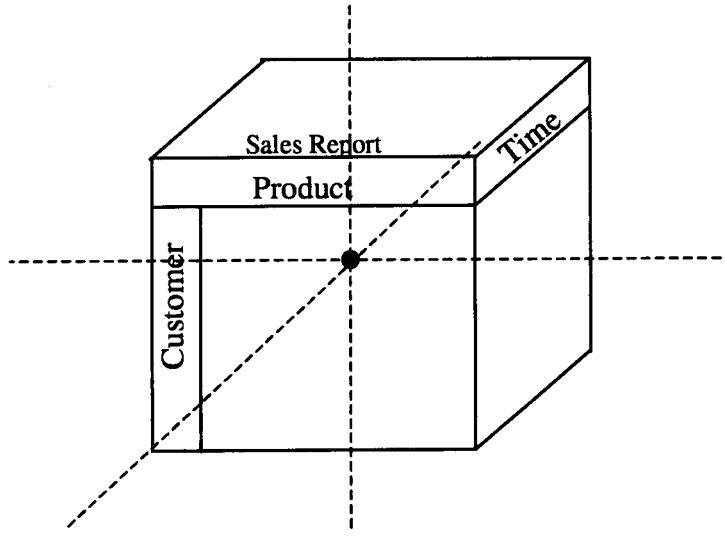


Figure 5.3 Example of a three dimensional cube

Now the dot represents some information about the sale of a particular product to a particular customer at a particular time. The information contained in the dot is still the same as before in that it is either atomic or complex and is usually numeric. All that has changed is that there are more dimensions by which the information represented by the dot may be analysed.

It follows, therefore, that the dot will continue to represent the same information irrespective of how many dimensions are needed to analyse and report upon it. However, it is not possible to represent more than three dimensions diagrammatically using this approach. In effect, the dot is 'trapped' inside this three dimensional diagram. In order to enable further dimensions of analysis to be represented diagrammatically, the dot must be removed to a different kind of structure where such constraints do not apply. This is the rationale behind the development of the Dot Modelling methodology. In Dot Modelling the dot is placed in the centre of the diagram and the dimensions are arranged around it as shown in Figure 5.4:

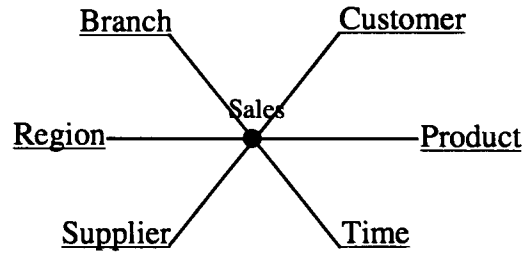


Figure 5.4 Simple multi-dimensional Dot model

The model readily adopts the well understood radial symmetry of the dimensional star schema.

5.3.2.2 The components of a Dot model

There are three basic components to a Dot Model diagram. They are:

1. **The Dot**. The Dot represents the facts. The name of the subject area of the dimensional model is applied to the facts. In the Wine Club, the facts are represented by 'sales'.

2. **Dimension names**. Each of the dimensions is shown on the model and is given a name.

3. **Connectors.** Connectors are placed between the facts and dimensions to show first level dimensions. Similarly, connectors are placed between dimensions and groupings to show the hierarchical structure

Emphasis has been placed on simplicity so there are virtually no notational rules on the main diagram. It is sensible to place the dot near the centre of the diagram and for the dimensions to radiate from the dot. This encourages a readable dimensional shape to emerge.

The Dot model for the Wine Club is reproduced in Figure 5.5:

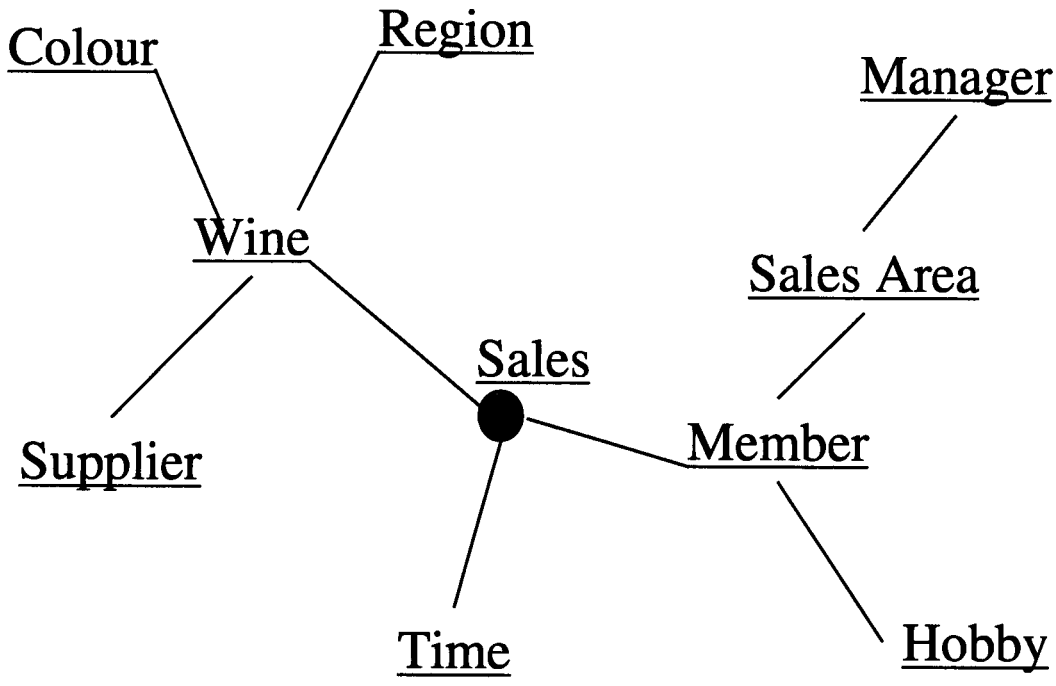


Figure 5.5 Representation of the Wine Club using a Dot model

The attributes for the facts and dimensions are not shown on the diagram. Attributes are described on supporting worksheets. Similarly, the temporal requirements are represented on supporting worksheets rather than on the diagram.

The method uses a set of worksheets. The worksheets are included in the appendices to this thesis. Some of the worksheets are completed during the conceptual design stage of the development and some are completed during the logical design stage.

The first worksheet is the data model worksheet itself. It contains the following:

1. The name of the application, or model. For example – The Wine Club – Sales

2. The diagram, as shown in Figure 5.5

3. A list of the 'fact' attributes (i.e. 'Quantity' and 'Value' in the Wine Club)

4. For each fact attribute, some information describing the fact is recorded under, what is commonly known as, 'Metadata'. Its purpose is to document the business definition of the

attribute. This is to solve the problem of different people, within an organisation, having differing views about the semantics of particular attributes. The descriptions should be phrased in business terms.

A second worksheet documents the dimensions. This part of the method holds some of the more complex information regarding the dimensional model. The model name is given on each page to ensure that parts of the document set are not mistakenly mixed-up with other models' documents. The purpose of the dimensions worksheet is to aid the designers of the system to understand the requirements in order to assist them in the logical design.

For each dimension the following items of information are recorded:

- The name of the dimension as it is understood by the business people. For example – 'Supplier'

- The retrospection of the dimension

- The existence attribute for the dimension. For dimensions with permanent retrospection, an example of which might be 'Region' in the Wine Club, there is no requirement to record the existence of a dimension because, once established, the dimension will exist as long as the database exists. With other dimensions, however, an attribute to represent existence would be needed so that, for instance, the Wine Club would be able to determine which wines were currently stocked.

- The frequency of the capture of changes to the existence of the dimension. This will help to establish whether the dimension will be subject to errors of temporal synchronisation.

For each dimension, a set of attributes is also defined on a separate worksheet. The existence attribute has already been described. The following description refers to the properties of other attributes. So for each attribute, the following information is recorded.

- The name of the dimension that 'owns' it.

- The name of the attribute. This is the name as the business (non-technical) people would refer to it.

- Retrospection. Whether or not the historical values of this attribute should be faithfully recorded.

- Frequency. This is the frequency with which the data is recorded in the data warehouse. This is an important component in the determination of the accuracy of the data warehouse.

- Dependency. This relates to causality and identifies other attributes that this attribute is dependent upon.

- Identifying attribute. This indicates whether the attribute is the identifying attribute, or whether it forms part of a composite identifying attribute.

- Metadata. A business description of the attribute.

- Source. This is a mapping back to the source system. It describes where the attribute actually comes from.

- Transformations. Any processing that must be applied to the attribute before it is eligible to be brought into the data warehouse. Examples of transformations are: the restructuring of dates to the same format, the substitution of default values in place of nulls or blanks.

- Data Type. This is the data type, and precision of the attribute.

Information about dimensional hierarchies is captured on the hierarchies worksheet. Pictorially, the worksheet shows the names of the higher and lower members of the hierarchy. The following information is also captured:

- The retrospection of the hierarchy

- The type of existence

- The frequency of capture

- Metadata describing the nature of the hierarchy.

5.4 Evaluation of the method

5.4.1 Assessment against requirements

The support for time is inherent in the Dot Modelling Methodology. The diagram, which presents a dimensional view of the subject area to be developed in the data warehouse, does not show how time is to be treated. In this sense, the diagram is a snapshot (Jensen et al (1994)) model. It is in the supporting information, which is captured on worksheets, where the issues surrounding time are dealt with. This is consistent with the conclusion that was drawn from the evaluation of other temporal methods in chapter three. By capturing the requirements for time on the supporting documentation, the diagram is able to show the business requirements and still retain the dimensional shape.

The diagram itself can be regarded as a simplified ER diagram. The simplification is made possible by taking account of the diagrammatic requirements for data warehouses and the points listed and described in section 5.3.1. The 'spreadsheet' analogy that was described in the introduction to Dot Modelling in section 5.3.2.1 has been found to be useful in aiding business people to understand the notation.

The requirements for the representation of time are captured using the concept of retrospection that was defined in section 5.2.3. One of three possible values for retrospection is recorded for each dimension, hierarchical relationship and attribute. In this way the method allows for time to be properly accounted for. Previous definitions have led to some confusion of practitioners and it is to be hoped that, by introducing a new and precise definition, the confusion will be removed. True retrospection enables time to be properly represented in dimensional hierarchies. In chapter two the absence of such a facility was identified as a major potential cause of inaccuracy in data warehouses.

The model also supports the recording of causality that was described in chapter two section four. The term causality has been used to define changes that occur in data warehouses which, in turn, cause other changes to occur. The synchronisation of the timing of such changes in a

data warehouse is important if accuracy is to be maintained, especially in cases where the warehouse is populated by various disparate operational source systems. In a Dot Model, dependencies between attributes can be recorded to alert the designers about the need to synchronise the capture of changes of some attributes.

Further to this point I have identified, in chapter two section five, the problem of the synchronisation of the timing of changes that occur in the operational systems and the data warehouse. Time delays between the capture of changes in the operational systems and the subsequent implementation of the changes into the data warehouse can now be accounted for by recording the frequency of the capture of each dimension and attribute.

The Dot Modelling method is, therefore, able to record all the requirements for the representation of time that have been identified in this thesis.

5.4.2 Practical experience

Since the method was originally developed, I have used it in telecommunications and financial services applications. As a result of my experiences I have developed a practitioner's guide (a copy of the guide is included in Appendix A). The guide provides some instructions for practitioners on the use of the method. It has been found that an approach involving two two-day workshops tends to produce good results, firstly with 'softer' issues involving the capture of business requirements and, secondly, with the capture of the temporal requirements.

The first workshop is called the 'Information Strategy' workshop and this is where the initial Dot Model is produced. Emphasis is placed on the business requirements and the creation of a dimensional model, by the business people, that reflects their view of their information requirements. The second workshop is called the 'Component Analysis' workshop and this is where the detailed requirements for each attribute are identified.

The guide has been reviewed by my employer's senior data warehousing practitioners and the feedback has been favourable. The world-wide data warehouse programme manager wrote:

'I finished reviewing your Dot Modelling Guide
and I think it is excellent!
I certainly hope that we continue to invest in the
creation of the rest of the guide.
I would love to see us structure a "dimensional modelling"
service around the dot modelling methodology'.

The method has been placed on my employer's internal web site and is available for use by any of the company's consultants, world wide, who has a need to conduct a requirements gathering exercise for a data warehouse project.

All the feedback I have received has been positive with constructive criticism being limited to detailed points such as the need to extend the length of the workshops when foreign language interpreters are needed. Although the guide requests feedback of any kind, the majority has been limited to conversational anecdotes. However, the following few sets of comments have been received. The first is from a consultant working on customer care data warehousing project for a retail bank. It has not been modified other than to correct typing errors:

I love the approach - I find that anything pictorial is easier for all to understand, & dot models are certainly pictorial.

I think the guide ought to have a "mgmt summary" at the front, giving an overview of the expected outcomes from the dot modelling workshop, & where it fits in the overall data gathering process (i.e.: what it does do, what it doesn't do).

I think it may also be worth having a high-level summary (table?) showing an agenda for the two days: Part of my problem was not having time to go thoroughly through the guide beforehand & so I was unaware that the methodology needed 2 workshops to capture first dimensions & secondly facts.

Each session in the guide should follow a similar format: e.g., Session title; estimated time; objectives; inputs; outputs, all prior to the description/method.

The workshop we ran at <customer name> - Business & User goals and info reqs. for two areas (Sales and Call Centre Statistics) - had to be completed in 1

day. It's all we could get!!. Not surprisingly, we failed to get all of the info we needed. We got goals, dimensions, & questions the participants needed to have answered by the "to-be" system (queries/reports) for sales, but failed to get the measures & facts. When we came to Call Centre Statistics, we didn't have time to go through the whole process again unfortunately. We resorted to discussions of what they have today & what they want in the future, followed up by individual interviews to flush out the gaps. Elegant, huh?

Business goals: We teased 6 of these out of the participants (3 business, 2 business/IT, 2 IT), of which one was deemed to be so far in the future it was beyond the scope. This probably took slightly more than the 30 mins suggested. Measures were at best woolly for most of these goals.

We didn't do the business strategy step: instead, we focused on the questions they wanted to ask & tied those back to the business goals (which I guess is the same). Most of the attendees were pretty switched-on so we got away with it! Had time permitted it would have been a good step, though.

The teaching of the dot model was a doddle. Everyone caught on really quickly (about 7 minutes). One break-out team managed to produce a dot model for each of the 5 business goals, the other team managed only 2 but in much greater depth & with some hierarchy defined already.

The dot models were recorded in Visio (file attached F.Y.I.), and are being pasted into the final report.

We never got into the Component Analysis stage, so I can't comment on experiences. My initial reaction on reading this was that the business types may get quite bored with this as it's quite "low-level" & IT-like. On the other hand, it's quite necessary. I'm not knocking it since I can't think up an alternative!

On the whole, I really wish we'd had time to cover the remaining steps in the

guide - it would have made the engagement much easier.

Congrats on work done: I'll use it more fully next time.

I was particularly pleased that the consultant was able to use the method without having studied it beforehand. I have used the method myself and have reached a similar conclusion based on my own experiences and on feedback received by other practitioners who have used it.

Another consultant, who now works for a completely different organisation, recently commented:

'Apart from a well known insurance company in 1997, I've used this technique in two telecomm companies - one mobile and one fixed - and seen it work in all three. The major reason to use the technique in the latter case was to check the validity of an existing ER model with the business (something which the existing DW consultants had paid scant attention to)'.

This next account arose following a conversation where the consultant stopped me to recount his experiences. I asked him to write them down and send them to me:

'During a recent engagement with a client a classic business and IT issue relating to language and misunderstanding occurred. Technical people with a deep routed understanding of dimensional models attempted in a workshop forum to capture requirements for a data warehouse from marketing and customer services people. The technical people used language and tools which left the audience confused and in a state of helplessness.

*In an attempt to recover the situation **'dot modelling'** was introduced. The front end of the process focuses on the capture of requirements using a pictorial representation of data and how items are linked to yield a result. The process was easy for the client's to understand and created a representation which they both helped create (indicating buy-in) and subsequently used as a communications tool.*

From a technical perspective the process was instrumental in performing the task of capturing the requirements. Beyond that it helped in creating the crucial link between business and IT which is at the route of the successful development of systems that contribute to business performance.

As a side effect 'dot modelling' also highlighted that in order to raise the probability that the implementation of a warehouse will be successful an approach radically different to traditional systems development is required'.

This rather succinct comment was received from a consultant from France.

'It is simple and can be understood by most business

and non-technical users;

I believe it is very appropriate to design data warehouses'

Finally, this feedback was received from a very experienced data warehouse practitioner:

'Given that the masters of DW do little to help us through the labyrinth of time I believe that your method provides a very powerful guide. As a practitioner I find the time aspect the most difficult to get my own head around let alone anyone else! The process you present gives me a lot to think about and provides me, as a consultant, with a framework to ensure

that I collect all the facts, avoiding quick decisions based purely on instinct'.

An interesting development occurred in Spring 1999 when working with a major telecommunications company in the UK. Their business objective is to build a 'customer centric' information data model that covers their entire enterprise. In dimensional modelling terms this means several dimensional models, each having a different subject area, that share the customer dimension During a workshop session with this customer I was able to show how the whole model might look using a single diagram, which I now refer to as 'Joining the Dots'. The diagram is shown in Figure 5.6 below:

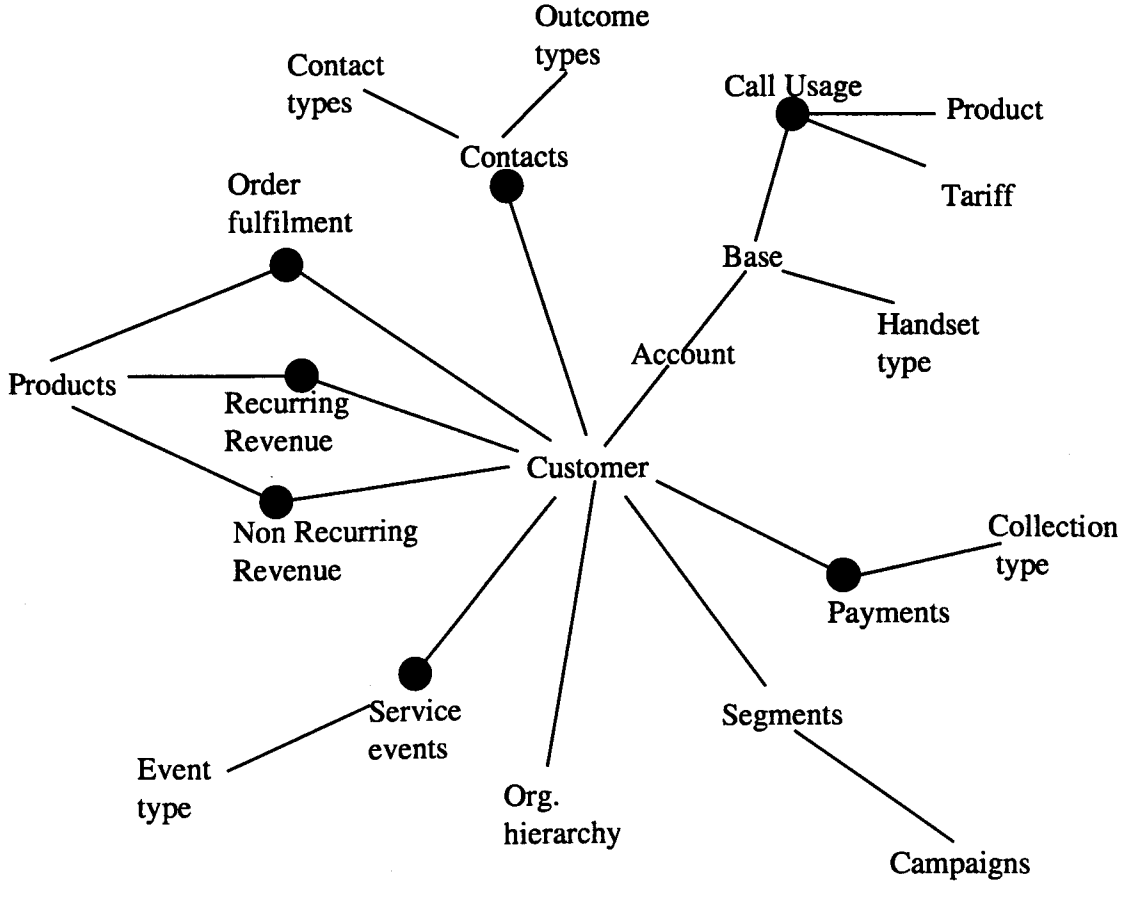


Figure 5.6 Customer centric Dot Model

Figure 5.6 shows seven separate dimensional models that share some dimensions. This illustrates that, even with very complex situations, it is still very easy to determine the individual dimensional models using the Dot Modelling notation because the radial shape of each individual model is still discernible.

The use of the Dot Model, in conjunction with business focused workshops (see Appendix A, the practitioners' guide) enables the 'softer' business requirements, in the form of business objectives or KPIs, to be expressed in information terms so that the data warehouse can be designed to provide precisely what the business managers need.

5.5 Summary of the requirements of a data warehouse model

This chapter has described the requirements of the conceptual model with respect to time. The principal requirement of the model is that it enables the complex temporal requirements for time to be fully recorded so that the main objective, which is the improvement of accuracy, can be achieved. I have shown that the Dot model enables these requirements to be recorded.

In summary, there are two main areas in which a data warehouse model can help with improving the accuracy of query results with respect to time:

1. The first of these refers to, what I have described as, the retrospection of each data warehouse object within the dimensional hierarchies. It is important that each dimension, relationship and dimensional attribute is assessed so that its retrospection requirements can be classified. The classification of retrospection has been fully described in this chapter.

2. The second area is the delay in time between the capture of changes to data (inserts, updates, deletes) in the operational system and the subsequent capture of those same changes, where applicable, in the data warehouse. The time lag was described fully in Chapter 2 Section 5.2 and can be the cause of permanent temporal inaccuracies in the data warehouse.

A further requirement is that the model should enable the business people to build the conceptual abstraction themselves. They should be able to construct the diagrams, debate them and replace them. I have shown through my own experiences, and those of other

consultants in the field, that business people have found Dot Models relatively easy to construct and use.

The next chapter deals with the implementation issues. I will show that a systematic approach can be applied to the implementation of retrospection using the concept of existence that was described in the conclusion to Chapter 3. Also in the next chapter, guidance is provided on ways in which the time lag problem can be resolved.

6 Implementation

6.1 Introduction

This chapter of the thesis deals with solutions to the implementation of the representation of time in data warehouses. Retrospection was introduced in the previous chapter and various options as to how it might be implemented are explored in this section. In connection with retrospection, the subject of 'existence', which was discussed in chapter three, is developed and used to show how some of the queries that were very difficult or not possible to express using the type two method in chapter four, can be written successfully.

Also in this chapter, the solution is developed further to show how it can be transformed into a relational logical model.

In practice, the logical modelling stage of the Conceptual→Logical→Physical progression is now usually omitted and designers will move from a conceptual model straight to a physical model. This involves writing the Data Definition Language (DDL) statements directly from the conceptual model. This practice has evolved over time because relational databases have become the assumed implementation platform for virtually all database applications. This applies to operational systems as well as informational systems such as data warehouses. However, many data warehouses are implemented not on relational database systems but on dimensional database systems that are generally known as On-Line Analytical Processing (OLAP) systems. There are currently no standards in place to assist designers in the production of logical models for OLAP proprietary database systems as there are for relational systems. Even so, in order to produce a practitioner's guide to developing data warehouses, the logical design process must pay some regard to non-relational database management systems. Where the target database management system is not relational, in the absence of any agreed standards, it is not possible to produce a physical model unless the designer has intimate knowledge of the proprietary DDL or physical implementation processes of the DBMS in question.

The temporal solutions that are put forward will have an impact on performance. The thesis briefly considers the ramifications of this and some solutions are suggested.

The chapter then provides recommendations as to the circumstances when each of the available temporal solutions should be adopted. Finally the chapter defines some constraints that have to be applied to ensure the implementation of the representation of time does not compromise the integrity of the data warehouse.

6.2 Logical Modelling

When reviewing these problems with the objective of trying to formulate potential solutions, the following are the main requirements that need to be satisfied.

1. Accurate reporting of the facts. It is very important that, whenever a fact entry is joined to a dimension entry, the fact ***must*** join to the correct dimension with respect to time. Where a snowflake schema exists, the fact entry must join to the correct dimension entry no matter where that dimensional entry appears in the hierarchy. In reality, it is recognised that this can never be wholly achieved in all cases as deficiencies in the capture of data from the operational systems of the organisation will impinge on our ability to satisfy this requirement. We can only hope to entirely satisfy this requirement insofar as the warehouse model itself is concerned and to minimise, as far as possible, the negative affect caused by the operational systems.

2. Accurate recording of the changes in dimensions to support dimension browsing. Dimension browsing, as has been shown, is a very significant component of the data warehouse usage. It is important to ensure that the periods of existence (validity) of dimensions, relationships and attributes are recorded accurately with respect to time where this has been identified as a business requirement. Again, the ability to do this is constrained by the accuracy and quality of data supplied by the operational systems. It is important that the warehouse does not compound the problem.

In keeping with the solutions put forward by others, and due to the lack of any standards in the OLAP model, our solutions will concentrate on the relational logical model.

6.3 The implementation of retrospection

6.3.1 Introduction

One of the principal requirements of a data warehouse model is the ability to enable the proper representation of time. The way in which I propose to achieve this is to use the retrospection concept that was developed in the previous chapter.

This section begins by stating a general rule about the use of time in data warehousing and goes on to put the previous work into context. It then explores an approach to the implementation of retrospection by the use of existence attributes.

In order to recognise that data warehouses are temporal applications and, therefore, the importance of time in data warehousing, the following general axiom is stated:

Every query executed in a data warehouse must relate to a time period.

If a query is executed that does not have an explicit time constraint, then the inferred time period is 'for all time'. Queries that embrace all of time insofar as the data warehouse is concerned, can be generally regarded as nonsensical because 'all of time' simply means the time that the database has been in existence. Whereas it may be sensible to aggregate across all products or all customers in order to ascertain some information about, say, total revenue for a period of time, it does not make sense to apply the same approach to the time dimension under normal circumstances. It is likely that readers of this thesis may be able to think of circumstances where the absence of a time constraint makes perfect sense, which is why the axiom is offered as a general rule rather than a fixed one. The principle still holds.

None of the work carried out so far explicitly provides support for time in the sense of true retrospection. Kimball's type two approach exists to ensure that each fact entry may be joined to a dimensional entry that accurately reflects the values of the attributes of that dimension when the fact was recorded. The method serves a valuable purpose but it is primarily concerned with consistency of attribute values, rather than time consistency.

Bair and Snodgrass describe different classes of tables and queries. They relate these temporal tables and queries to the grand temporal debate but do not offer any recommendations as to how support for time should be implemented in a data warehouse application.

The time dimension, when used in a dimensional model, serves merely as a means of grouping the facts together in various ways. The time dimension is seen by practitioners as the data warehouse object that provides the 'time varying' characteristics of data warehouse applications.

So at the end of all the research and investigation, very little guidance as to how to solve the problems surrounding time in data warehousing has been forthcoming. The only method that partially solves the problems is Kimball's type two which, while it does provide some benefits, has some drawbacks which have been described. Other methods for representing time have never been applied to data warehousing.

6.3.2 The use of existence attributes

Following the description of existence, which was described in the conclusion to chapter three, I propose that temporal support for each element, where required, within the dimensional structure of a data warehouse should be implemented by an existence attribute. Various ways of representing such an existence attribute may be considered. At the simplest level for an entity, an existence attribute could be added to record whether each occurrence is currently active or not. However, if full support for history is needed, then this requires a composite existence attribute consisting of a start time and an end time that records each period of existence. Some special value should be assigned to the end time to denote a currently active period. Each element may have such an existence attribute implemented as follows:

1. For the temporal support of an entity that may have a discontinuous existence, a separate table is required, consisting of the primary key of the dimension table and an existence period. If multiple existence is not possible, the existence period may be added to the dimension table.

2. For the temporal support of a relationship between dimensions, a separate table is required, consisting of the primary keys of both participating dimensions together with an existence period.

3. For the temporal support of an attribute of a dimension, a separate table is required, consisting of the primary key of the dimension, an existence period and the attribute value.

It should be noted that the concept of existence attributes is not new. In fact the approach could be described as a kind of 'selective' attribute timestamping. Attribute timestamping has been described in many papers and is mentioned in chapter three of this thesis during the review of the TEER^2 method (see Lai et al (1994)). However, use of attribute timestamping has largely been limited to research into temporal database management systems and has not before been associated with data warehousing. It is the selective adoption of attribute timestamps, in the form of existence attributes, for data warehousing purposes that is new.

The use of existence attributes solves the problem of cascaded extraneous inserts into the database caused by the use of Kimball's type two solution with a slowly changing hierarchy that was described in chapter four. The reason is that there is no need to introduce a generalised key for a dimension, because changes to an attribute are kept in a separate table. However, the performance and usability of the data warehouse needs to be considered. I can show by example that it does add to the complexity so I would propose that it should be allowed only for those elements where there is a clearly identified need for temporal support. It is for the decision makers to choose which elements are sufficiently important as to justify such treatment.

To explore the benefits of existence attributes, let us consider that the user wishes to know the number of customers that a particular sales executive is responsible for. The intuitive query, reproduced from Figure 4.16 in the review of logical models chapter, is shown in Figure 6.1:

```
Select count(*)
        from Sales_Exec S,Customer C
        where S. SalesExecNum=C. SalesExecNum
        And S.Name = 'Tom Sawyer'
```

Figure 6.1 Non expert query to count customers

If type two is implemented, the result would definitely be wrong. Each customer would have one or more rows in the table depending on the number of changes that had occurred to the customer's record. The result would substantially overstate the real situation.

Type two works by creating a new dimension record with a different generalised key. One simple solution is that the dimension is given an additional attribute that signifies whether the

row is the latest row for the customer. The value of this attribute declares the existence of the dimension and remains true until the row is superseded. In its simplest form the existence of a dimension can be implemented using a Boolean attribute. This means that when a change is implemented, the previous latest row is updated and the existence attribute is set to 'false'. The new row has its existence attribute set to 'true'.

The new query to determine the number of customers, that a particular sales executive is responsible for, is shown in Figure 6.2:

```
Select count(*)
    from Sales_Exec S,Customer C
    where S. SalesExecNum=C. SalesExecNum
    And S.Name = 'Tom Sawyer'
    And C.Existence = true
```

Figure 6.2: Use of a simple existence attribute

The end user query software could be configured to add this last line to all dimension tables so the user need not be aware of it. However, this does not entirely solve the problem because it is not answering the question: How many customers is Tom Sawyer responsible for now? Rather, it is answering the question: How many customers has Tom Sawyer ever been responsible for? One method toward solving this problem would be to also set the existence attribute to false when the customer ceased to be a customer. Whether or not this is possible depends on the ability of the data warehouse processing to detect that a customer's status had become 'inactive'. For instance, if the customer's record in the source system were to be deleted, the data warehouse processing could infer that the customer's existence attribute in the data warehouse should be updated to false.

Another variation is to make use of Null values in the existence attribute, as follows:

1. Null for not existing

2. Not null for existing (i.e. current[5]).

If the column was called 'existence', to identify all customers who were still active customers, then the query would be:

```
Select count(existence)
    from Sales_Exec S,Customer C
    where S. SalesExecNum=C. SalesExecNum
    And S.Name = 'Tom Sawyer'
```

Figure 6.3 Use of a simple existence attribute in the 'Count' built in function

Due to the way nulls are treated (i.e. they are ignored unless explicitly coded for), this expression of the query is almost as simple as the original, intuitive query phrased by the user in Figure 6.1.

Furthermore, if the *'true'* value of 'existence' was 1, then the query in Figure 6.4 would also return the correct result:

```
Select sum(existence)
    from Sales_Exec S,Customer C
    where S. SalesExecNum=C. SalesExecNum
    And S.Name = 'Tom Sawyer'
```

Figure 6.4 Use of a simple existence attribute in the 'Sum' built in function

This appears to effectively solve the problem of determining who the current customers are. Thus even the simplest existence attribute can improve Kimball's original type two method considerably.

The existence attribute could be implemented using a single 'effective date'. This has the advantage that we can determine when a change occurred. However, such a method does not

[5] The representation of 'Now' and the semantics associated with the current time are discussed as a temporal database issue by Clifford et al (1997). A detailed discussion of this subject is interesting and some attention must be paid to it while non temporal databases are being used.

allow us to answer the previous query (How many customers is Tom Sawyer responsible for?) because, again, there is no means to determine inactive customers. The use of row time stamping, using a pair of dates, does enable the question to be answered so long as the end date is updated when the customer becomes inactive. However there are many questions, such as state duration and transition detection questions, that are very difficult to express and even some that are impossible to express using this approach.

As an example I will now describe a business problem that is particularly relevant at the present time. Telecommunication companies frequently measure their performance by the amount they have been able to increase customer retention and, in doing so, decrease customer defection to other companies. The telecommunications industry's vernacular term for such defection is called 'churn' and the reduction of churn has become a major business objective. The concept of churn is rapidly spreading to other types of organisation that have large numbers of customers and lots of competition such as banks, supermarkets and utilities and its adoption as a measure of customer satisfaction is becoming widespread. It is a problem that could equally apply to the Wine Club. It is now becoming a common practice to contact customers, that have 'churned', in sales campaigns in an attempt to attract them back. Where organisations are successful in doing this, the customers are reinstated with their previous identifiers so that their previous history is available to the customer care staff. The incidence of discontinuous existences is, therefore, becoming more common.

The need to monitor churn and to establish the reasons for it tends to create a need for queries that return results in the form of a time series. The following exemplify the type of questions we would like to express:

1. How many members did we lose during the last quarter of 1998, compared to 1997 and 1996? The result of such a query would be a time series containing three periods and a number attached to each period. This is an example of a temporal selection query.

2. Of the members that were lost, how many had been members continuously for at least one year? The loss of long standing members might be considered to be as a result of worsening service. The result from this question is also a time series but the query contains an examination of durations of time. So this query is a state duration query.

3. How many of these members experienced a change of administration because they moved address in the year that they left? Perhaps they are unhappy with the level of service provided by the new area. This question is concerned with the existence of the relationship between the member and the sales area. It is also an example of a transition detection query.

4. How many price changes did they experience in the year that they left?. Perhaps they were unhappy with the number of price rises imposed. This is similar to question three in that it is a transition detection query but it is applied to the value of an attribute, in this case the selling price of a bottle of wine, instead of a relationship.

These requirements cannot be satisfied using a single Boolean attribute to describe the existence of a dimension as there is a requirement to make comparisons between dates. Neither can the queries be expressed using a single date attribute for the reason previously explained. It seems clear that the expression of such queries requires a composite existence attribute that is, logically, a period comprising a start time and an end time.

It has been shown that row timestamping can provide a solution in many cases, but not all, and the resulting queries are complex to write. A simpler solution is sought.

The approach to be adopted will be to use a separate existence attribute, in the form of a composite start time and end time, for each dimension, relationship and attribute where retrospection has been defined to be 'true'. It is assumed that these existence attributes will be held in separate tables. So the existence attribute for Members' existence will be held in a table called 'MemberExist' and the existence attribute for the period during which a member lives in a sales area will be called 'MemberSalesAreaExist'.

Using the composite existence attribute, the first query can be expressed as in Figure 6.5:

```
select 'Q4 1998' as quarter, count(*)
        from MemberExist me
        where me.ExistenceEnd between '1998/10/01' and '1998/12/31'
        union
select 'Q4 1997' as quarter, count(*)
        from MemberExist me
        where me.ExistenceEnd between '1997/10/01' and '1997/12/31'
union
select 'Q4 1996' as quarter, count(*)
        from MemberExist me
        where me.ExistenceEnd between '1996/10/01' and '1996/12/31'
```

Figure 6.5 Count of members that have left

The query in Figure 6.5 could have been answered using row time stamping only if the end time stamp was updated to show that the member was no longer active. The distinction is made between the existence attribute and the row timestamp because the existence attribute is a single purpose attribute that purely records the existence of the member. The row time stamp is, as has been stated, a multi purpose attribute that records other types of changes as well. In order to express the query using row time stamps, it would have to be written as a correlated subquery to ensure that only the latest record for the member was evaluated. This means that discontinuous existences could not be detected.

The second query can be expressed as in Figure 6.6:

```
select 'Q4 1998' as quarter, count(*)
        from MemberExist me
        where me.ExistenceEnd between '1998/10/01' and '1998/12/31'
        and (me.ExistenceEnd - me.ExistenceStart) > 365
union
select 'Q4 1997' as quarter, count(*)
        from MemberExist me
        where me.ExistenceEnd between '1997/10/01' and '1997/12/31'
        and (me.ExistenceEnd - me.ExistenceStart) > 365
union
select 'Q4 1996' as quarter, count(*)
        from MemberExist me
        where me.ExistenceEnd between '1996/10/01' and '1996/12/31'
        and (me.ExistenceEnd - me.ExistenceStart) > 365
```

Figure 6.6 Count of long standing members lost

In order to express the third query, it is assumed that there is a separate existence attribute for the relationship between the member and the sales area. This is shown in Figure 6.7:

```
select 'Q4 1998' as quarter, count(*)
        from MemberExist me, MemberSalesAreaExist msa
        where me.ExistenceEnd between '1998/10/01' and '1998/12/31'
        and (me.ExistenceEnd - me.ExistenceStart) > 365
        and me.MemberCode = msa.MemberCode
        and msa.ExistenceStart between '1998/01/01' and '1998/12/31'
union
select 'Q4 1997' as quarter, count(*)
        from MemberExist me, MemberSalesAreaExist msa
        where me.ExistenceEnd between '1997/10/01' and '1997/12/31'
        and (me.ExistenceEnd - me.ExistenceStart) > 365
        and me.MemberCode = msa.MemberCode
        and msa.ExistenceStart between '1997/01/01' and '1997/12/31'
union
select 'Q4 1996' as quarter, count(*)
        from MemberExist me, MemberSalesAreaExist msa
        where me.ExistenceEnd between '1996/10/01' and '1996/12/31'
        and (me.ExistenceEnd - me.ExistenceStart) > 365
        and me.MemberCode = msa.MemberCode
        and msa.ExistenceStart between '1996/01/01' and '1996/12/31'
```

Figure 6.7 Lost members that moved house

The query in Figure 6.7 is an example of a combined state duration and transition detection query.

As with the other queries, in order to express the fourth query, it is assumed that there is a separate existence attribute for the bottle price:

```
select 'Q4 1998' as quarter, me.MemberCode, count(distinct spe.WineCode)
        from MemberExist me, SalesPriceExist spe, Sales s, Time t
        where me.ExistenceEnd between '1998/10/01' and '1998/12/31'
        and (me.ExistenceEnd - me.ExistenceStart) > 365
        and me.MemberCode = s.MemberCode
        and s.WineCode = spe.WineCode
        and s.TimeCode = t.TimeCode
        and t.year = 1998
        and spe.ExistenceStart between '1998/01/01' and '1998/12/31'
        group by quarter, me.MemberCode
        having count(distinct spe.WineCode) > 5
union
select 'Q4 1997' as quarter, me.MemberCode, count(distinct spe.WineCode)
        from MemberExist me, SalesPriceExist spe, Sales s, Time t
        where me.ExistenceEnd between '1997/10/01' and '1997/12/31'
        and (me.ExistenceEnd - me.ExistenceStart) > 365
        and me.MemberCode = s.MemberCode
        and s.WineCode = spe.WineCode
        and s.TimeCode = t.TimeCode
        and t.year = 1997
        and spe.ExistenceStart between '1997/01/01' and '1997/12/31'
        group by quarter, me.MemberCode
        having count(distinct spe.WineCode) > 5
union
select 'Q4 1996' as quarter, me.MemberCode, count(distinct spe.WineCode)
        from MemberExist me, SalesPriceExist spe, Sales s, Time t
        where me.ExistenceEnd between '1996/10/01' and '1996/12/31'
        and (me.ExistenceEnd - me.ExistenceStart) > 365
        and me.MemberCode = s.MemberCode
        and s.WineCode = spe.WineCode
        and s.TimeCode = t.TimeCode
        and t.year = 1996
```

and spe.ExistenceStart between '1996/01/01' and '1996/12/31'
group by quarter, me.MemberCode
having count(distinct spe.WineCode) > 5

Figure 6.8 Lost members affected by price increases

The query in Figure 6.8 shows members that left the club in the last quarter of the year and that had experienced more than five price changes during the year. This is another example of a combined state duration and transition detection query.

In dealing with the issue of churn, this approach of trying to detect patterns of behaviour of customers is typical.

It is accepted that the queries have drawbacks in that they are quite complex and would prove difficult for the average data warehouse user to write. Also, each query actually consists of a set of smaller queries and each of the smaller queries is responsible for processing a discrete point in time or a discrete duration. Any requirement to increase the overall time span, or to break the query into smaller discrete timespans would result in many more queries being added to the set. So the queries cannot be generalised to deal with a wide range of times. In section 3.3 of this chapter I explore and propose an approach for making the queries much easier to express and generalise.

In this section I have shown that the use of existence periods do provide a practical solution to the implementation of true retrospection. This enables time to be properly represented in the dimensional structures of the data warehouse and satisfies one of the major requirements that was described in Chapter 5 Section 5.

6.3.3 The use of the time dimension

In this section of the thesis I explore how the use of the composite existence attribute that was introduced in the previous section, together with the time dimension, may allow the expression of some complex queries to be simplified.

The purpose of the time dimension is to provide a mechanism for constraining and grouping the facts, as do the other dimensions in the model. This thesis is examining methods for providing support for time in the dimensions and dimensional hierarchies as well as the facts.

The four queries listed in Figures 6.5 to 6.8 show that similar time constraints apply to the dimensions as to the facts. Therefore it seems appropriate to allow the users of the data warehouse to express time constraints on dimensions using the same approach as they do with the facts.

Snodgrass (1997) says that joins are not allowed between dimensions but he does not say why this is so. Kimball (1997b) also proscribes the use of the time dimension with other dimensions because he is concerned that the semantics of time in the dimensions is different to that of facts and is potentially misleading. He again uses the 'supplies from stock' example as his reason. The view that the time dimension should not be used in dimensional browse queries is supported implicitly by the conventional star schema and snowflake schema data models that show the time dimension as being related to the fact table alone. There is no relationship between the time dimension and any other dimension on any dimensional model that I have seen.

In considering this matter, two points emerge. Firstly, the time dimension provides a simple interface to users when formulating queries. Preventing them from using the time dimension with other dimensions means that the users will be able to place time constraints by selecting terms such as '2nd Quarter 1998' in queries involving the fact table but not the dimensions. In dimensional queries, the explicit time values have to be coded. Secondly, some dimensional browsing queries are much easier to express if a join is permitted between the dimension in question and the time dimension. Further, I have discovered that some useful but complex queries, such as those in the previous section, can be generalised if a join to the time dimension is permitted. This is described below.

Referring to the queries expressed in Figures 6.5 to 6.8, the first query (Figure 6.5) is: How many members did we lose during the last quarter of 1998, compared to 1997 and 1996?. Using the time dimension, it can be expressed as shown in Figure 6.9:

```
select t.Quarter, count(*)
    from MemberExist me, Time t
    where me.ExistenceEnd = t.TimeCode
    and t.Quarter in ('Q41998', 'Q41997', 'Q41996')
    group by t.Quarter
```

Figure 6.9 Count of members lost using the time dimension

Changes to the temporal scope of the query can be effected simply by altering one line of the predicate instead of creating additional discrete queries.

The query from Figure 6.6 is: Of the members that were lost, how many had been members continuously for at least one year? This can be expressed as follows:

```
select t.Quarter, count(*)
    from MemberExist me, Time t
    where me.ExistenceEnd = t.TimeCode
    and t.Quarter in ('Q41998', 'Q41997', 'Q41996')
    and (me.ExistenceEnd - me.ExistenceStart) > 365
    group by t.Quarter
```

Figure 6.10 Count of long standing members lost, using the time dimension

The third query, from Figure 6.7, is: How many of the members experienced a change of administration because they moved address in the year that they left? Using the same assumptions as before, the query can be expressed as in Figure 6.11:

```
select t1.Quarter, count(*)
    from MemberExist me, MemberSalesAreaExist msa, Time t1, Time t2
    where me.ExistenceEnd = t1.TimeCode
    and t1.Quarter in ('Q41998', 'Q41997', 'Q41996')
    and (me.ExistenceEnd - me.ExistenceStart) > 365
    and me.MemberCode = msa.MemberCode
    and msa.ExistenceStart = t2.TimeCode
    and t2.Year = t1.Year
    group by t1.Quarter
```

Figure 6.11 Lost members that moved house, using the time dimension

Finally, the fourth query, from Figure 6.8, is: How many price changes did they experience in the year that they left? This can be expressed as shown in Figure 6.12:

```
select t1.Quarter, me.MemberCode count(distinct spe.WineCode)
    from MemberExist me, SalesPriceExist spe, Sales s,
    Time t1, Time t2, Time t3
    where me.ExistenceEnd = t1.TimeCode
    and t1.Quarter in ('Q41998', 'Q41997', 'Q41996')
    and (me.ExistenceEnd - me.ExistenceStart) > 365
    and me.MemberCode = s.MemberCode
    and s.WineCode = spe.WineCode
    and spe.ExistenceStart = t2.TimeCode
    and s.TimeCode = t3.TimeCode
    and t2.Year = t1.Year
    and t3.Year = t2.Year
    group by t1.Quarter, me.MemberCode
    having count(distinct spe.WineCode) > 5
```

Figure 6.12 Lost members affected by price increases, using the time dimension

Thus I conclude that allowing the time dimension to be joined to other dimensions, when existence attributes are used, enables a simpler expression of some temporal queries

In order to adopt a change of approach whereby joins are allowed between the Time dimension and other dimensions, we have to alter the data model. There now exists a relationship between the Time dimension and some of the other dimensions.

Only those dimensions that need true support for time will be related to the Time dimension. Part of the Wine Club model has been reproduced in Figure 6.13:

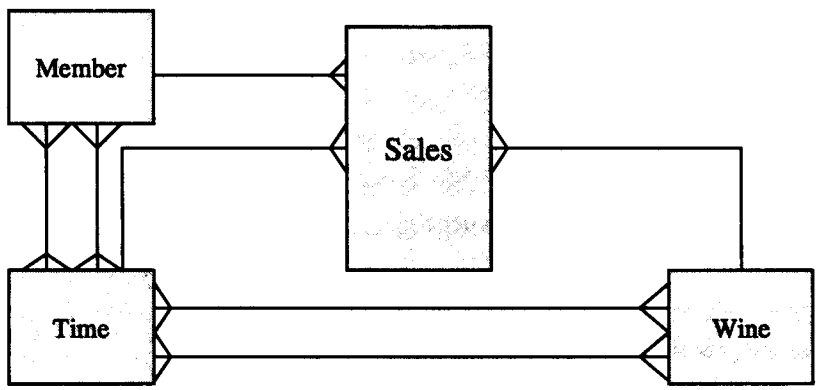


Figure 6.13 ER diagram showing new relationships to the time dimension

Figure 6.13 shows Time having a relationship with Sales, as before, and also with the Member and Wine dimensions. In fact, as Figure 6.13 shows, there are two relationships between the time dimension and other dimensions, one for the start time of a period and one for the end time.

A problem caused by this change in the conventional approach is that the model will immediately lose its simple star schema shape. This is one of the requirements placed on other conceptual models during the review. In creating this new idea, I have introduced a further problem to be solved. It is important that the dimensional shape is not lost.

The solution could be that the Time dimension is removed altogether from the model. It has been said before that the Time dimension is always included as part of a dimensional model because time is always a dimension of analysis in a database that records history. In further recognition of the fact the data warehouses are temporal databases, the explicit inclusion of a

time dimension could be regarded as unnecessary. So, on the assumption that the Time dimension is a 'given' requirement, I have adopted the view that its inclusion is implicit. This means that the diagram does not need to model the Time dimension explicitly. However, this causes a problem with the entity-relationship modelling methodology in that it would be misleading to have implicit entities that are deemed to exist but are excluded from the diagram. The Dot modelling methodology does not have this limitation and will be adapted to accommodate the new requirement.

However, the Time dimension does have attributes that are specific to each application and so it is not something that can be ignored altogether. For instance, data warehouses in some types of organisation require specific information about time, such as:

- half-day closing
- prevailing weather conditions
- effect of late opening due to staff training
- whether the store was open for 24 hours

This type of information cannot be obtained through any type of derivation. So there is a need for some means of specifying the attributes for time on a per-application basis. In the Dot modelling methodology I propose to solve this problem by the introduction of a table that will satisfy the requirements previously handled by the explicit time dimension as well as the requirements covered in this section.

The table could be given a standard table name for use in all applications. The use of 'Time' as a name for the table is likely to conflict with some RDBMS reserved word list so, for the purposes of this thesis, the name 'dot_time' will be use to describe the table.

Each application will have its own requirements as to the columnar content of the dot_time table although some columns, such as the following, would almost always be required:

- Date

- Day name
- Week Number
- Month name
- Month Number
- Quarter
- Year

Practitioners could add value to their customers by bringing a 'starter' dot_time table that might contain, say, ten years of history and ten years of future dates. This seems like a large amount of data but, in reality it is less than eight thousand rows where the granularity is daily. For finer levels of granularity e.g. seconds, it is sensible to provide two time tables. The first contains all the days required, as before, and the other would contain an entry for each second of a single day (i.e. from 00:00:00 to 23:59:59). It is then a simple matter to join to one table, in the case of dimensional changes, or both tables in the case of, say, telephone calls. In this way, multiple grains of time can be accommodated.

Practitioners could also provide standard programmed procedures, perhaps using the 'user defined functions' (see Visser(1998)) capability that is available as an extension to some RDBMS products, to add optional columns such as weekends and bank holidays etc. although some customisation of the dot_time table is almost inevitable for every application.

The removal of the explicit time dimension from the conceptual model to the logical model is a step forward in the approach to the design of data warehouses. It also goes some way to recognising that data warehouses are true temporal applications and the support for time is implicit in the solution, rather than having to be made explicit on the data model.

6.3.4 Logical schema

The implementation of true retrospection for any dimension, dimensional attribute or relationship requires that the existence of the lifespan, of the object in question, is recorded. This must be in the form of a period marking the starting and ending times. The introduction

of such periods changes the structure of the dimension. For instance, the Member dimension from the Wine Club model has a requirement for true retrospection on its own existence as well as the relationship with the Sales Area dimension and the Members' addresses. Each of these would be given their own existence period attributes. A diagram depicting the logical model for the Wine Club is shown in figure 6.14 below. The diagram shows how the implementation of true retrospection using existence attributes results in the creation of new relations:

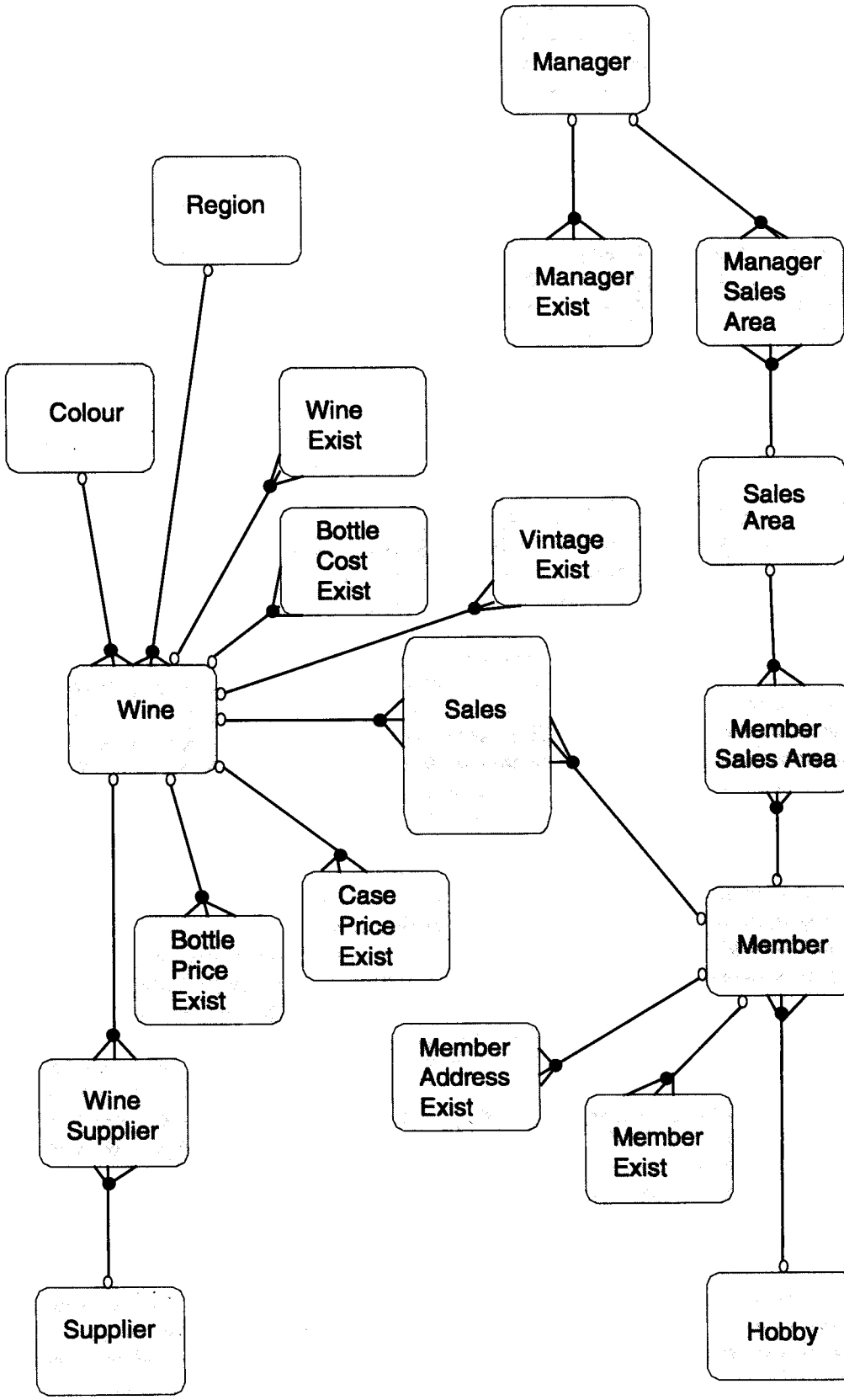


Figure 6.14: Logical model of the Wine Club

Each of the relations is now described:

Relation Member
Member_Code
Member_Name
Hobby_Code
Date_Joined
Primary Key (Member_Code)
Foreign Key (Hobby_Code references Hobby.Hobby_Code)

Relation Member Exist
Member_Code
Member_Exist_Start
Member_Exist_End
Primary Key (Member_Code, Member_Exist_Start)
Foreign Key (Member_Code references Member.Member_Code)

Relation Member Address Exist
Member_Code
Member_Address_Exist_Start
Member_Address_Exist_End
Member_Address
Primary Key (Member_Code, Member_Address_Exist_Start)
Foreign Key (Member_Code references Member.Member_Code)

Relation Member Sales Area
Member_Code
Sales_Area_Code
Sales_Area_Code_Exist_Start
Sales_Area_Code_Exist_End
Primary Key (Member_Code,Sales_Area_Code,Sales_Area_Code_Exist_Start)
Foreign Key (Sales_Area_Code references Sales_Area.Sales_Area_Code)
Foreign Key (Member_Code references Member.Member_Code)

Relation Sales_Area
Sales_Area_Code
Sales_Area_Name
Primary Key (Sales_Area_Code)

Relation Manager Sales Area
Manager_Code
Sales_Area_Code
Manager_Sales_Area_Code_Exist_Start
Manager_Sales_Area_Code_Exist_End
Primary Key (Manager_Code,Sales_Area_Code,Manager_Sales_Area_Code_Exist_Start)
Foreign Key (Sales_Area_Code references Sales_Area.Sales_Area_Code)
Foreign Key (Manager_Code references Manager.Manager_Code

Relation Manager
Manager_Code
Manager_Name
Primary Key (Manager_code)

Relation Manager Exist
Manager_Code
Manager_Exist_Start
Manager_Exist_End
Primary Key (Manager_Code, Manager_Exist_Start)
Foreign Key (Manager_Code references Manager.Manager_Code)

Relation Colour
Colour_Code
Colour
Primary Key (Colour_Code)

Relation Hobby
Hobby_Code
Hobby_Name
Primary Key (Hobby_Code)

Relation Region
Region_Code
Region_Name
Country
Primary Key (Region_Code)

Relation Sales
Member_Code
Wine_Code
Time_Code
Quantity
Value
Foreign Key (Member_Code references Member.Member_Code)
Foreign Key (Wine_Code references Wine.Wine_Code)

Relation Supplier
Supplier_Code
Supplier_Exist
Supplier_Name
Supplier_Address
Phone
Primary Key (Supplier_Code)

Relation Wine
Wine_Code

Wine_Name
Colour_Code
ABV
Region_Code
Primary Key (Wine_Code)
Foreign Key (Colour_Code references Colour. Colour _Code)
Foreign Key (Region _Code references Region. Region _Code)

Relation Wine_Exist
Wine_Code
Wine_Exist_Start
Wine_Exist_End
Primary Key (Wine_Code, Wine_Exist_Start)
Foreign Key (Wine_Code references Wine.Wine_Code)

Relation Bottle_Price_Exist
Wine_Code
Bottle_Price_Exist_Start
Bottle_Price_Exist_End
Bottle_Price
Primary Key (Wine_Code, Bottle_Price_Exist_Start)
Foreign Key (Wine_Code references Wine.Wine_Code)

Relation Case_Price_Exist
Wine_Code
Case_Price_Exist_Start
Case_Price_Exist_End
Case_Price
Primary Key (Wine_Code, Case_Price_Exist_Start)
Foreign Key (Wine_Code references Wine.Wine_Code)

Relation Bottle_Cost_Exist
Wine_Code
Bottle_Cost_Exist_Start
Bottle_Cost_Exist_End
Bottle_Cost
Primary Key (Wine_Code, Bottle_Cost_Exist_Start)
Foreign Key (Wine_Code references Wine.Wine_Code)

Relation Vintage_Exist
Wine_Code
Vintage_Exist_Start
Vintage_Exist_End
Vintage
Primary Key (Wine_Code, Vintage_Exist_Start)
Foreign Key (Wine_Code references Wine.Wine_Code)

Relation Wine_Supplier

Wine_Code
Supplier_Code
Wine_Supplier_Exist_Start
Wine_Supplier _Exist_End
Primary Key (Wine_Code, Supplier_Code, Wine_Supplier _Exist_Start)
Foreign Key (Wine_Code references Wine.Wine_Code)
Foreign Key (Supplier _Code references Supplier. Supplier _Code)

The logical schema has been included as an aid to clarity and is not intended to prescribe a physical model. Performance issues will have to be considered when implementing the solution and some denormalisation of the above schema may be appropriate. However, the above schema does represent examples of all three types of retrospection.

True retrospection of a dimension is exemplified in the existence of the member by providing a separate relation containing just the member code and the existence period. True retrospection of an attribute is shown by the 'Member Address' relation and true retrospection for a relationship is shown by the 'Member Sales Area' relation that records the relationship between the member and the sales area. In each case the primary key includes the start time of the period attribute.

False retrospection of a dimension existence is illustrated in the Supplier relation where the existence attribute is included within the relation itself. False retrospection is supported for the relationship between members and hobbies by the inclusion of the hobby code as a foreign key in the member relation. The member name, for example, also has the property of false retrospection.

6.3.5 Performance considerations

This thesis is concerned with accuracy of information and not with performance. However, it is recognised that performance considerations have to be made at some point and I am concerned that performance may be considered to be more important than accuracy when design decisions are taken. For this reason I will briefly raise the subject of performance.

A query involving the fact table such as 'The sum of sales for 1997 grouped by Sales Area' using a type two approach, with surrogate keys, would be expressed as shown in Figure 6.15:

```
Select SalesAreaCode, sum(s.value)
    From sales s, member m, time t
    Where s.MemberSurrogate = m.MemberSurrogate
    And s.timecode = t.timecode
    And t.year = 1997
    Group by SalesAreaCode
```

Figure 6.15 Sales value by sales area code using type two

Using the normalised schema the same query requires a more complex join, as the query in Figure 6.16 shows:

```
Select SalesAreaCode, sum(s.Value)
    From sales s, MemberSalesArea msa, time t
    Where s.member code = msa.member code
    And s.time code = t.time code
    And t.time code between msa.start and msa.end
    And t.year = 1997
    Group by SalesAreaCode
```

Figure 6.16 Sales value by sales area code using existence attributes

The join between the 'time' and 'member sales area' dimensions is not a natural join and is unlikely to be handled efficiently by most RDBMS query optimisers. A practical solution to solving the performance issue in these cases, while retaining the benefit of existence attributes, is to copy the sales area code to the fact table. As has been stated previously, the attributes of the fact table always have the property of 'permanent' retrospection. So the accuracy of the results would not be compromised and performance would be improved considerably, even better than the original because a whole table is omitted from the join. This is shown by the query in Figure 6.17:

```
Select SalesAreaCode, sum(s.Value)
    From sales s, time t
    where s.time code = t.time code
    And t.year = 1997
    Group by SalesAreaCode
```

Figure 6.17 Refined query to obtain sales value by sales area code using existence attributes

The relationship between the member and sales area dimensions must be left intact in order to facilitate dimensional browsing. This is because the decomposition of the hierarchy is not reversible. In other words the hierarchy cannot be reconstructed from the fact table since, if there is no sale, there is no relationship between a member and a sales area. So it would not be a 'non-loss' decomposition.

The model enables dimensions to be queried and for time series types of analysis to be performed by the use of the time dimension that enables the sales area and member dimensions to be grouped by any time attribute.

Queries involving dimensions only, such as counting the number of members by year, would be expressed as in Figure 6.18:

```
Select year,count(*)
    From MemberExistence me, time t
    Where t.time code between me.start_date and me.end_date
    And t.timecode in (1998/12/31, 1997/12/31, 1996/12/31)
    Group by year
```

Figure 6.18 Counting members at points in time using the existence table

This appears to be a reasonably efficient query. At this point it is worth remembering that dimensional browse queries represent about eighty percent of queries executed on the data warehouse.

At this level in the model, there is no reason to distinguish between the requirements of false and permanent retrospection. The main reason for these classifications is concerned with

issue of population of the data warehouse and the capture of changed data values. In any case, the method handles these requirements quite naturally.

6.3.6 Choosing a solution

The choice as to which solution is most appropriate depends upon the requirements and the type of data warehouse object under consideration.

Retrospection provides a mechanism for freeing the dimensions to be regarded as a source of information in their own capacity. If there is a requirement for true historical analysis to be performed on any components of the dimensional structure, then those components will have a classification for retrospection of 'true'. If there is a requirement to perform state duration queries or transition detection queries then the most appropriate choice is to provide an existence attribute, in the form of a start time and an end time, that is applied to the object in question. As far as dimensions are concerned, this is the main reason for allocating a true retrospection. At any point in time, a member will or will not be active due to the discontinuous nature of their existence. True retrospection for dimensions results from the need for time series analysis of the existence of dimensions so the existence attribute approach is the only choice in this case.

False retrospection for dimensions is similar. It enables the currently existing and no longer existing dimensions to be identified. So it is possible to determine which members currently exist and which do not. It is not possible to carry out any form of time based analysis. The best approach for implementing false retrospection for dimensions is to use an existence attribute, containing true or false values, as described earlier in this chapter.

Permanent retrospection requires no support at all as the existence of the dimension is considered never to change.

Support for relationships that implement dimensional hierarchies is entirely missing from the type two solution and any attempt to introduce a type two or a row timestamp solution into a relationship may result in very large numbers of extraneous cascaded inserts as has been shown in the previous chapter. It is recommended that use of these techniques is avoided where there are implicit (as implemented by a star) or explicit (as implemented by a snowflake)

hierarchies unless all the superordinate objects in the hierarchy have the property of false or permanent retrospection.

When retrospection is true, if state duration or transition detection analysis of the relationship is required, the relationship must be implemented using an existence attribute with a start time and an end time. There is a slight difference between the existence of relationships and the existence of dimensions. The subordinate member will always be engaged in a relationship with a superordinate sales area because the participation condition for the member is mandatory. It is not, therefore, a question of existence versus non existence, rather a question of changing the relationship from one sales area to another and the existences pertaining to relationship instances. The discontinuous nature still exists but applies to changing relationships rather than gaps in durations. A row timestamp approach can be used where these requirements do not apply as long as the cascade insertion problem is manageable.

False retrospection in relationships is best implemented by allowing the foreign key attribute to be overwritten in the subordinate dimension's row. No special treatment is required. With permanent retrospection, the foreign key attribute is not expected to change at all, so no special treatment is required.

The situation with regard to attributes is similar again. If state duration or transition detection analysis is required then the simplest and most flexible solution is to use an existence attribute with a start time and an end time. Where these types of analysis are not needed, the use of row timestamps can be considered as long the problem relating to cascade insertions is manageable. False retrospection in attributes requires no special support as the attribute can be safely overwritten. Permanent retrospection also requires no support as the attribute will not change.

In the practitioners guide, the logical modelling section will need to provide a simplified approach for practitioners to follow and so the table in Figure 6.19 has been produced.

<table>
<tr><td></td><td colspan="2">Dimension</td><td colspan="3">Relationships</td><td colspan="2">Attributes</td></tr>
<tr><td rowspan="5">True</td><td colspan="2">State duration or transition detection analysis required ?</td><td colspan="3">State duration or transition detection analysis required ?</td><td colspan="2">State duration or transition detection analysis required ?</td></tr>
<tr><td>Yes</td><td>No</td><td>Yes</td><td colspan="2">No</td><td>Yes</td><td>No</td></tr>
<tr><td>Existence Period</td><td rowspan="3">Row Time stamp</td><td rowspan="3">Existence Period</td><td colspan="2">Does the Hierarchy Change Regularly?</td><td rowspan="3">Existence Period</td><td>Is this attribute at the lowest level in the hierarchy?</td></tr>
<tr><td></td><td>Yes</td><td>No</td><td></td><td>Yes</td><td>No</td></tr>
<tr><td></td><td>Existence Period</td><td>Row Time stamp</td><td></td><td>Row Time stamp</td><td>Existence Period</td></tr>
<tr><td>False</td><td colspan="2">Existence true/false attribute updated using type 1 method</td><td colspan="3">Type 1</td><td colspan="2">Type 1</td></tr>
<tr><td>Perm</td><td colspan="2">Will not change</td><td colspan="3">Will not change</td><td colspan="2">Will not change</td></tr>
</table>

Figure 6.19 Choosing a solution

The purpose of the table is to provide a simplified guide to aid practitioners in the selection of the most appropriate solution for the representation of time in the dimensional structure.

6.4 Frequency of changed data capture

In the pursuit of accuracy relating to time, we need to know whether the data we are receiving for placement into the data warehouse accurately reflects the time that the change actually occurred. This was identified as another source of inaccuracy with respect to the

representation of time in the summary of requirements in Chapter 5 Section 5. This requirement cannot be fully satisfied by the data warehouse in isolation, as is now described.

So far as the facts are concerned, the time that is recorded against the event would be regarded, by the organisation, as the valid time of the event. That means it truly reflects the time the sale occurred, or the call was made.

With the dimensions, we are interested in capturing changes. As has been discussed, changes to some attributes are more important than others with respect to time. For the most important changes, we will expect to record the times that the changes occurred. Some systems are able to provide valid time changes to the dimensions but most are not equipped to do this. So we are faced with the problem of deducing changes by some kind of comparison process that periodically examines current values and compares them to previous values to determine precisely what has changed and how.

The only class of time available to us in this scenario is transaction time. Under normal circumstances, the transaction time is the time that the change is recorded in the operational system. Often, however, the transaction time changes are not actually recorded anywhere by the application. Changes to, say, a customer's address simply result in the old address being replaced by the new address, with no record being kept as to when the change was implemented. Other systems attempting to detect the change, by file comparison method, have no real way of knowing when the change occurred in the real world or when it was recorded into the system.

So, in a data warehouse environment, there are two time lags to be considered. The first is the lag between the time the change occurred in the real world, the valid time, and the time the change is recorded in an operational system, the transaction time. Usually, the organisation is unaware of the valid time of a change event. In any case, the valid time is rarely recorded. The second time lag is the time it takes for a change, once it has been recorded in the operational system, to find its way into the data warehouse.

There is a school of thought that states that there should be a delay in the transfer of data to the data warehouse. According to Inmon (1992), *at least* twenty four hours should be allowed to pass, after a change to a dimensional attribute is known to the operational system, before

the change is recorded in the data warehouse. He calls this 'putting a wrinkle in the data'. He cites three reasons for doing this:

1. Keeping the data warehouse up to date with the operational systems is expensive.

2. Having a twenty four hour delay allows the data to 'settle'.

3. There is no temptation to attempt to do operational processing in the data warehouse, or warehouse processing in the operational environment.

Inmon is either failing to recognise the issue surrounding the inaccuracy resulting from this approach or is assuming that the level of inaccuracy is not sufficiently significant to be worthy of mention.

Devlin (1997) recognises the problem and observes that the delayed capture technique enables only an approximation of historical data to be built. There is very little reference to this issue in the literature other than the requirement for 'non-volatility'. Stedman (1996) observes that if the results keep changing, then it can become difficult to accurately assess what is happening. There is a difference between the continual update of the data warehouse with the new fact data and the need to keep the dimensional data up to date as it changes. The distinction appears not to have been recognised by many of the authors.

The solution is to minimise the time lags inherent in this process. Although that is often easier said than done, the objective of the designers must be to identify and process changes as quickly as possible so that the temporal aspect of the facts and dimensions can be synchronised.

6.5 Constraints

6.5.1 Double Counting constraints

Double counting occurs when the joining of tables returns more rows than should be returned. This problem is usually avoided if the general rules about the structure of dimensional models are followed.

However, the introduction of existence attributes into the dimensional model increases the risk of error by changing the nature of the relationships between dimensions with other dimensions, or dimensions with facts, from simple (1:n) to complex (m:n).

The problem is best described by the use of an example:

The following is an example of a sale of wine. Figure 6.20 shows the bottle cost existence:

Wine Code	Start	End	Bottle Cost
4504	1996/02/27	1997/03/31	4.36
4504	1997/03/31	Now	4.79

Figure 6.20 Example of the existence of a wine cost entity

The bottle cost has an existence attribute. The existence is continuous but a change in the cost price of a bottle of the wine has occurred. This has resulted in the generation of a new row.

Figure 6.21 shows a fragment of the Sales fact table detailing a sale of the wine above:

Wine Code	Day	Quantity	Value
4504	1997/03/31	5	24.95

Figure 6.21 A single sale of wine

The query in Figure 6.22 is executed, which is intended to show the sales value and costs:

```
Select w.wine_name, s.value "Revenue", sum(s.quantity * w.bottle_cost) "Cost"
    from Sales s, Wine w, BottleCostExistence bce
    where w.wine_code = s.wine_code
    and w.wine_code = bce.wine_code
    and s.day between bce.start_date and bce.end_date
    group by w.wine_name
```

Figure 6.22 Sales of wine query using existence attributes

The result set in Figure 6.23 is returned:

Wine Name	Revenue	Cost
Chianti Classico	24.95	21.80
Chianti Classico	24.95	23.95

Figure 6.23 Example of double counting

The result is that the sale has been double counted. The problem has occurred because of an overlap of dates that caused the Sale, which was made on the 31st March 1997, to successfully join to two rows. The date of the change may well be right. The old cost price ceased to be effective on the 31st March 1997 and the new price took effect immediately, on the same day.

As far as the query processing is concerned, the multiple join is also correct. The join criteria have been met.

What is wrong is the granularity of time. There is an implicit constraint that states that: *Time overlaps in existence are not permitted in a dimensional model.* Therefore, if the dates are both correct, then the granularity is incorrect and a finer grain, such as time of day, must be used instead.

The alternative is to ensure that the end of the old period and the start of the new period actually *'meet'*. This means that there is no overlap, as shown in Figure 6.24:

Wine Code	Start	End	Bottle Cost
4504	1996/02/27	1997/03/30	4.36
4504	1997/03/31	Now	4.79

Figure 6.24 Wine cost entity showing no overlaps in existence

It is equally important to ensure that no gaps are inadvertently introduced, as in the Figure 6.25:

Wine Code	Start	End	Bottle Cost
4504	1996/02/27	1997/03/30	4.36
4504	1997/04/01	Now	4.79

Figure 6.25 Wine cost entity showing gaps in existence

If the data in the table in Figure 6.25 was used, then the result set would be empty.

The query used in the example was phrased so as to aid clarity. It is worth remembering that in data warehousing most queries, involving joins to the fact table, use arithmetical functions to aggregate the results. The chances of users identifying errors from the result sets of such queries are seriously reduced when large numbers of rows are aggregated.

Therefore, in order for the concept of existence to work, temporal constraints must prevent any form of overlapping to occur on periods otherwise there is a risk that double counting might occur. The following query, using the temporal construct 'overlaps', would detect such an occurrence.

```
Select R1.PK
    From R1, R2
    where R1.PK = R2.PK
    and R1.Period <> R2.Period
    where R1.Period overlaps R2.Period
```

Figure 6.26 Constraint ensuring no overlaps in existence

R1 and R2, in Figure 6.26, are synonyms for the same relation and PK is the primary key. The relation is subjected to a self join in order to identify temporal overlaps.

This query can be rewritten, without using the temporal constructs, as in Figure 6.27:

```
Select R1.PK
    from R1, R2
    where R1.PK = R2.PK
    and (R1.Start <> R2.Start or R1.End <> R2.End)
    and R1.Start <= R2.End
    and R2.Start <= R1.End
```

Figure 6.27 Constraint ensuring no overlaps in existence using standard SQL

The periodic execution of such queries would enable the detection of double counting errors.

6.5.2 Referential Integrity Constraints

Several axiomatic referential integrity constraints can be specified:

- The period of existence of the subordinate object must be contained within a single period of existence of the superordinate object.

- In a dimensional hierarchy, as long as the subordinate exists, there must also exist a relationship between the subordinate dimension and its superordinate dimension. This is due to the mandatory participation condition relating to subordinate dimensions in dimensional models.

- During the period of existence of a dimension, all the attributes of the dimension must exist. It follows, therefore, that gaps in existence of attributes are not allowed while the dimension is in existence.

There are some constraints relating to retrospection, as follows:

If a dimension has true retrospection:

- It can have attributes that have true retrospection. The lifespans of those attributes must fall within the lifespan periods of the dimension. If the dimension ceases to exist, then the attributes with true retrospection should cease to exist at the same time.

- It can have attributes that have false retrospection. These attributes can change only when the existence attribute for the dimension indicates that the dimension exists. They cannot change when the existence attribute for the dimension indicates that the dimension does not exist.

- It can have attributes that have a value, for retrospection, of permanent as these values never change.

If a dimension has false retrospection:

- It can have attributes that have true retrospection. If the dimension ceases to exist, then the attributes with true retrospection should cease to exist at the same time. When the dimension changes from non-existent to existent, these attributes should begin a new interval of existence at the same time.

- It can have attributes that have false retrospection. These attributes can change only when the existence attribute for the dimension indicates that the dimension exists. They cannot change when the existence attribute for the dimension indicates that the dimension does not exist.

- It can have attributes that have permanent retrospection as these values never change.

If a dimension has a value for retrospection of permanent, there are no constraints that restrict the kind of attributes the dimension can have.

Inmon (1998) makes a point that, over time some constraints within a table may change as the business rules change. An example from the Wine Club might be that, before 1998, a wine may be supplied by exactly one supplier at a time whereas, after that, a wine may be supplied by one or many suppliers. Inmon calls this 'bounded' referential integrity.

6.5.3 Deletion constraints

It is for the owners of the data warehouse to determine the values of retrospection that are to be applied to each data warehouse object. The three approaches place very different requirements on the design of the warehouse and the accuracy of the results from queries will vary depending on the choices made. Some rules can be applied:

The rules governing referential integrity violations in relational databases with respect to deletions must be applied to existence. Where a dimension changes its status from being existing to becoming logically non-existing, this is equivalent to the dimension being logically deleted. The application of the rules has to be applied selectively.

- *Cascade* delete cannot be used because this would result in the deletion of facts and this would have the effect of invalidating the database. Although the data warehouse would retain integrity in the sense that the references would remain intact, the database would return incorrect results.

- *Nullifying* the references (effectively – deleting the relationship) cannot be used because the participation condition of the dimensions, which would be nullified, is mandatory. If it were permitted to delete the relationships, queries that aggregated all the facts using one

dimension would produce different totals to other dimensions. This would have the effect of invalidating the results.

- The only method for dealing with changes to existence is to use the *Restricted* effect. This means that when a dimension's existence ceases any referencing dimensions must have their relationship existences closed, and new relationship existences created that refer to an existing dimension, before the previous dimension is allowed to have its existence terminated.

6.6 Evaluation and summary of the logical model

This chapter of the thesis has focused on the implementation of the solutions that were introduced into the conceptual model in chapter five.

Principally, chapter six has focused on the implementation of retrospection. I have shown how true retrospection can be implemented by the use of existence attributes and how the use of such attributes enables queries to be expressed that are otherwise very difficult to write and impossible to generalise using the approach adopted generally by practitioners. Existence attributes can be described as selective attribute timestamps that are applied only where needed. Such an approach has not before been applied to a data warehousing solution.

The pursuit of accuracy due to the proper representation of time in data warehouses is a major objective of this research and the implementation of true retrospection has enabled a very much greater level of accuracy to be achieved in queries involving dimensions as well as maintaining accuracy in queries involving facts. It is recognised that there are some problems with time that are beyond the capability of the data warehouse to resolve in that they lie in the operational systems.

As a result of attempting to simplify the expression of queries involving time, I have discovered that the use of the time dimension can be greatly extended to enable joins to dimensions rather than just to facts. This approach challenges standard practices and is not recommended by some authors but the advantages that accrue, in terms of the simplification of queries, are significant enough to justify its adoption. In introducing this new idea I have created another problem, which is that the new relationships tend to obliterate the essential star shape of the dimensional model because the time dimension is now related not only to the

fact table but also to one or more of the dimensions and even some attributes. Upon reflection however, in recognition of the fact that data warehouses are temporal databases, it seems entirely appropriate to change the status of the time dimension from being an explicit component of the data model to becoming an implicit component that does not, therefore, need to exist on the diagram. Far from being simply a convenient way of removing the problem, I believe it is a benefit of the method and a natural consequence of accepting the fact that data warehouses are temporal databases.

In conclusion, the implementation of retrospection, together with the consideration given to simplification of queries, performance and temporal integrity constraints enables time to be properly represented in data warehousing. Additionally, the practitioner's guide provides assistance to data warehouse practitioners in enabling them to make the most appropriate choice of solutions depending on the circumstances of their application.

7 Conclusion

This final chapter of the thesis describes the contribution. The results of the research are summarised and subsequently discussed. Also, future research directions are considered.

7.1 Contribution

To begin with, however, I would like to list what I believe to be my contribution to the subject. My contribution is in several areas:

1. I have shown that existing methods do not properly represent time in data warehouses. I carried out a detailed critique of current approaches and encountered several flaws that are fully described in chapters three and four of this thesis.

2. I have created a conceptual model for data warehouses where none exists at present. This model is capable of fully supporting the temporal requirements. The model also supports the dimensional approach that is favoured by most practitioners. Dot modelling is described in chapter five and a full method for using it is included in the Appendix A.

3. Using the term 'retrospection', I have established precise definitions of the various types of requirement with respect to time that have until now been inconsistent. Using a systematic argument in chapter five (section 2.3) and chapter six (section 3) I have shown how retrospection, when implemented using existence attributes, enables the proper representation of time in data warehouses. Further, the method enables some dimensional queries to be expressed that are not possible using existing methods.

4. Current methods place constraints on the usage of the time dimension. In challenging these constraints in chapter six (section 3.3), I have been able to greatly simplify the expression of some important dimensional queries.

5. I have created a practitioner's guide to the representation of time in data warehouses that encompasses these features. The practitioner's guide builds on the theoretical method described in chapter five and is shown in Appendix A.

7.2 Summary

The principal subject of this thesis is time and the way that it is used in data warehousing applications. The original hypothesis that was put forward in chapter one is that the representation of time in data warehouses is generally not adequate and this can lead to inaccuracy in the results obtained from queries.

The presence of time and the dependence upon it is one of the things that sets data warehouse applications apart from traditional systems. Most business applications are suited to operating in the present environment where time does not represent a constraint. In many cases, dates are no more than descriptive attributes. In a data warehouse, time affects the very structure of the application. In a temporal database management system, support for time should be implicit in the structural fabric of the database. The absence of such a facility means that the data warehouse database, and the supporting application software, has to be made aware of time. The support for time has to be explicitly coded into the table structures and the queries.

The purpose of time in data warehousing is that it enables historical data to be held and queried upon. This means that users of data warehouses can view aspects of their enterprise at any specific point in time, or over any period of time, for which the historical data is recorded. This further enables the observation of patterns of behaviour over time so that we can make comparisons between similar or dissimilar periods e.g. this year vs. last year, seasonal trends. Armed with this information, we can extrapolate with the use of predictive models to assist us with planning and forecasting. We are, in effect, using the past to attempt to predict the future: for example, the use of information from past events and trends is commonplace in economic forecasting, social trend forecasting and even weather forecasting.

In a dimensional data warehouse application, we need to provide support for time in two areas, the subject area data (the facts) and the dimensional data. There is a need for a methodology to assist designers of data warehouses to deal appropriately with the requirements relating to time throughout the data warehouse development life cycle.

The approach adopted in this thesis fell into two main themes. The first theme covered conceptual modelling and reviewed methods of recording the temporal requirements of data

warehouse applications. The research focused on seven candidate methods that were intended to provide explicit support for time.

Most of the methods involved models based on the traditional entity relationship (ER) approach or the extended entity relationship (EER) approach. The tendency was to add further symbols, or to redefine the meaning of existing symbols to enable them to adopt a temporal interpretation. Some of the methods strayed outside the pure conceptual model into the domain of the logical model. All the methods reviewed tended to be more complex, rather than less complex, than the original ER and EER methods. The resulting diagrams were very cluttered and the dimensional shape quickly became difficult to discern. Also, the extra capabilities of the models introduced further notational rules that made the methods more difficult to use. The models were further complicated by the inclusion on the diagram of all the data objects together with their temporal modifications. The developers of the models wanted to describe the temporal requirements on the diagram but this caused a dilemma because it would have been inconsistent to include the temporal support for entities and relationships but not for attributes. I concluded that if the model was truly temporal then the support for time should be implicit and resolved that the diagram should show just the snapshot requirements of the facts and dimensions and the temporal requirements for all data objects would be removed to supporting documentation. The final solution is a much simplified adapted form of an ER model for dimensional modelling that is very appropriate for data warehousing. The method is called Dot Modelling. It has very few symbols and, consequently, very few notational rules. Support for time in Dot Modelling is provided using a new concept called 'Retrospection' that was introduced to enable a precise meaning to be attached to the temporal requirements of individual data warehouse objects. The retrospection of an object can be carried right through to implementation.

The second theme concentrated on logical modelling and how the temporal requirements that were recorded in the conceptual modelling stage could be implemented using the relational model as the target implementation platform. It was very difficult to find any real support for time outside of the temporal database field of research. There were two papers that helped to provide a mapping from the temporal definitions to a relational implementation. Other assistance came from established practitioners in the field of data warehousing. Support for time in existing methods was found to be limited to the facts, with only secondary

consideration given to the temporal requirements of the dimensions. Where support for dimensions was provided, it was found that some classes of temporal queries could not be expressed using the standard query language. Also, hierarchies in dimensions were completely ignored and it was found that their introduction can cause the method to become unwieldy.

In order to provide a better solution, the focus returned to retrospection and possible methods of implementation. As a result the concept of *existence* was developed that enables the three types of retrospection to be implemented. Using true retrospection and existence attributes it was found that the questions, that were previously not answerable, could now be answered. Existence attributes are implemented in a similar way to attribute timestamps but selectively, as required to support the temporal requirements of the business. Although the concept of attribute timestamping is not a new idea, its application in data warehousing *is* new.

A further development was that, by widening the use of the time dimension to allow implicit relationships between dimensions, some queries involving time in dimensions were found to be very much easier to express. This led to the reassessment of the role of the time dimension in the model from being explicit to adopting a more implicit position. This, it was reasoned, is more in keeping with the temporal nature of data warehouses and enabled the diagram to be significantly simplified.

A practitioner's guide has been produced that incorporates all these new features and which, it is hoped, will assist practitioners in the future development of dimensional data warehouses.

7.3 Discussion

In reflecting upon the research I have conducted and the thesis that has developed from this, it is my view that the field of data warehousing, especially dimensional data warehousing, is a new type of database that deserves to be accorded its own theory and its own set of practices. Dimensional systems have existed for many years but it is only recently that they have become popular and that any attempt to formalise their development has been made. Data warehouses have been popularised by a few industry visionaries, all of which have been cited in this thesis. Not surprisingly, there is a small contingent who claim to have been 'doing it for years, long before it was called data warehousing'. There exists a need to treat data warehouse database development in a different way to other kinds of database development. The paper 'Research

problems in data warehousing' (Stanford University Data Warehouse Project (Widom(1995))) described several problem areas for research including the problems surrounding time, changed data capture and data integration issues.

A very small number of practitioners such as W H Inmon and, most notably, R Kimball have attempted to provide an end to end approach to designing and building data warehouses. Kimball is the most influential in the field of design because his methods are far more detailed than those of any other. In trying to cover the whole subject, he and others appear to have overlooked some of problems at the detailed level. Kimball is credited with having invented the dimensional 'star schema' even though some would, again, say they had been 'doing it for years, long before it was called 'star schema' and has been given widespread support for this approach throughout the industry. Although the subject of time is mentioned quite frequently, it is never really dealt with in a satisfactory way in any of the literature.

My objectives are twofold. Firstly, there is a need to place time, and the temporal issues surrounding data warehousing, at the front of the minds of practitioners who are developing and implementing data warehouses in the field. In order to achieve this I have tried to augment and formalise the methods that we have been given which, until now, have been developed and implemented in a somewhat ad hoc fashion. The distinction between the conceptual model and the logical model has become rather nebulous over time and the development process suffers from this. This was very apparent when reviewing some of the temporal data modelling methodologies. Some features, of what should have been a conceptual model, were present for relational database performance reasons rather than for the capture of information requirements. Much of the literature regarded the conceptual star schema and the ultimate implementation model as the same thing. I mentioned earlier the need for a dimensional database theory, as distinct from, say, relational theory and there is hope in the shape of OLAP and the OLAP Council that one day some standard method may be agreed upon. When that happens we will need a traditional approach consisting of conceptual, logical and physical modelling. In order for such a methodology to be successful, it must take account of the temporal issues, as well as other issues that are outside the scope of this thesis. E F Codd et al (1993) attempted to introduce a set of rules to which all OLAP products would need to adhere in order to be classified as a 'true' OLAP product. The rule set

was widely discredited due to the fact that the paper, in which they were presented, was commissioned by an OLAP software vendor who just happened to satisfy all the criteria.

In conceptual modelling, there is always a conflict between the need to capture information requirements at a level that is usable by those who understand the information and the need to capture technical information about the information itself. Any attempt to satisfy both requirements on the same model will always compromise the model to the extent that it will satisfy neither of the requirements adequately. In this respect the Dot Modelling method does add some value in that, firstly, it has been used in real projects and found to be very easy for non-technical people to understand and use. Secondly, it does enable the temporal requirements to be recorded without making the main descriptive model any more complex because these requirements are moved to another section of the model. This provides for a link to the next stage of the process which is the production of the logical model. Most of the work previously undertaken, in data warehousing specifically, has been at the logical and physical level although, as has been said, the boundaries between the three levels have become increasingly indistinct. None of the established practitioners or researchers was able to express a view as to how temporal requirements for a dimensional data warehouse could be accommodated in current environments. Neither did any recognise the risks to accuracy of information resulting from inadequate attention to time. Almost no-one has acknowledged the fact that data warehouses are temporal applications with virtually no temporal support. This thesis has provided a link between a conceptual model that recognises temporal issues and a set of logical model guidelines that advise on how those requirements can be accommodated in a relational environment. Additionally, a practitioner's guide has been included that instructs practitioners, in their own language, how to assess the temporal needs of the application and how to implement them, given the tools at their disposal today. As a result of this thesis, people in the field can now be made aware of the temporal issues facing them and the consequential compromise to accuracy that they are imposing on their customers by paying insufficient attention to this problem. They can determine the requirements, classify them into discrete types, assess the affect of one changing data element on others and recommend an approach to implementation. There is no equivalent approach in the field at the present time.

The second objective is to raise the level of awareness of data warehousing within the research community. This point was first raised in the introduction to the thesis. There are other fields of interest that are loosely connected to data warehousing and that have generated some activity by researchers. One of these is data mining which is an application that uses large volumes of data. Fayyed et al (1996) describes data mining as 'determining patterns from observed data'. The research interest in data mining is in the techniques for pattern matching and how the mining algorithms might be designed to detect potentially interesting patterns of behaviour. Often, a data mining product will use a data warehouse database as the platform upon which to perform its investigation. So the quality of the information uncovered by a data mining application is dependent on the quality of information held in the data warehouse. If the data warehouse contains information that is inaccurate due to, say, temporal reasons, then the quality of information provided by data mining products, or any other application, is equally suspect.

Currently there is no proper dimensional database theory. I would like to see some development in this area and I will expand on this later. Also at present, there are no rules to follow except those that have become de facto standards. One of these is that all dimensional models should have 'time' as one of the dimensions. Kimball (1996a) says that the time dimension can be virtually guaranteed to be present in every data warehouse. The purpose of the time dimension is to segment the facts over time. The star schema is always drawn with a time dimension joined to the fact table, just like any other dimension. So there now exists a de facto 'law' that places a time dimension on the model. The result of my research has been to conclude that the time dimension not only has an explicit connection to the fact table, it has implicit links to any table, relationship and attribute that requires proper support for time. Allowing the join of the time dimension to other data warehouse objects enables some queries to be executed much more simply than was the case without it. This I found to be an exciting and unexpected discovery because it challenges the accepted way of doing things. The result is that the time dimension should become an implicit feature of the model rather than an explicit dimension although, obviously, it has to be implemented as an explicit table in order for the queries to work properly. However, retaining it on the diagram as a dimension would mean cluttering the model with extraneous connecting lines and would defeat one of the principal strengths of Dot Modelling that is its inherent simplicity. It also recognises the temporal

nature of the application by making the assumption that time is always a feature of dimensional data warehouses. So this is another impact of the thesis on the accepted standard way of conducting data warehouse design.

7.4 Future Research

It is hoped that this thesis may encourage other research into this field and serve as a platform from which new research is launched. There are several areas that would benefit from some attention by the research community.

Insofar as my own work is concerned, I have been focused entirely on data warehousing but I believe the concept of retrospection may be appropriate for more general use and I would encourage further research in this area. Although data warehousing is probably the largest application that needs support for time, there are others, for instance: the real time monitoring equipment in hospitals that need to take measurements of, say, heart rates at frequent time intervals. These systems do record history and a modelling methodology that uses retrospection may provide some benefit in this type of application.

It is also my view that further research should be undertaken into the field of temporal data modelling. I feel sure that making time an implicit feature of the model and dealing with it in the supporting documentation has significant advantages over the models that were reviewed as part of my research. Time should be regarded as an implicit feature of temporally aware databases and data models.

This thesis has only briefly touched on the issue of performance and I am sure that there is scope for research in this area. The practitioner's guide is an evolving document and I plan to add chapters to it over time. To be able to include a section on performance issues, based on research, would very much appreciated by practitioners.

A further useful facility in data warehousing would be the introduction of a method that enabled the estimation of the margin of error in a query due to the inadequate representation of time. Such a facility would give users of data warehouses a greater level of confidence in their query results. This is an area in which I am already engaged in research.

As previously mentioned, it would be useful if work were to be carried out into the development of a dimensional theory for the implementation of dimensional databases. All known database products which, collectively, fall under the banner of OLAP databases are, in fact, dimensional in nature. These products are quite numerous although the market is going through some upheaval at the present time due to the intervention of Microsoft. Nevertheless, the deployment of OLAP products as platforms for data warehouses is growing very quickly. OLAP products have some distinct advantages in the field of data warehousing because they are able to perform far more complex calculations and provide more sophisticated presentation of results than ever could be achieved using SQL. This shortcoming in the capability of SQL has been exposed by Kimball (1994). In a later paper Kimball (1996h) recommends some changes to compensate for the shortcomings. OLAP products are not constrained by standards in the same way as SQL and so are able to solve some of the problems.

There is a body that is attempting to develop a set of agreed standards for OLAP systems. The OLAP council was set up to co-ordinate the development of OLAP products and to enable some standards to be established. These objectives have not been met largely because of the conflicting interests of the OLAP product vendors from which the council is comprised. Some research in this area, led by an independent body, may help to bring them together again so that practitioners may have some solid foundation upon which to build the industry. Relational theory is now well established and we are a long way from any sort of agreement about temporal database theory. In any case there are no real temporal database management systems available for general use and, at the current time, there are none in prospect. The proliferation of dimensional models, together with the fact that they have been in existence for many years and taking the future prospects of OLAP into account tends to point to a greater need for some formalisation of the development of dimensional databases.

This thesis has touched upon some of differences between the relational approach and the dimensional approach but there are others. For instance, the security requirements are entirely different. There are no insert, update and delete issues, and consequently no requirements to 'commit' or 'rollback' transactions, in data warehousing because it is the loading process that performs the insertion tasks. There is no requirement to be able to grant these capabilities to end users. Instead there are security issues surrounding who is actually allowed to see which

information. A single fact table may contain facts for several product lines where some of the users are permitted to see only their own products. In a relational database, this has to be implemented using views with every user potentially having to use their own view. Where third party products are used as 'front end' query tools, this sometimes becomes a management headache. The dimensional theory would have its own security model and some research into this area would be helpful.

This thesis has tended to concentrate on the data warehouse database itself, with some attention being paid to the identification of changes in operational data and the capture of those changes in a way that minimises inaccuracy due to time lags. There are other areas in data warehousing that would benefit from being the subject of research. One of the subjects that has received significant attention from product vendors is what is commonly referred to as *metadata*. Due to the fact that data is extracted from disparate source systems and passes through transformation processes, including format integration and semantic integration, before being placed into the warehouse, the result may appear to be quite different to what the users would normally expect. As a result, there is a need for information to be held about the semantics of the data that the users can query and analyse before actually asking questions of the data warehouse itself. This is one form of metadata. According to Hurwitz (1997), there are three main forms of metadata. The other two are: the actual business semantic metadata that describe each data element from the business person's perspective and the navigational metadata, that software products need to assist them in their operation, such as the schema tables in relational databases. The subject of metadata is another area where the industry has tried and failed to agree on a standard approach. As the debate is taken forward, notice should be taken regarding the temporal aspects of metadata. Atre (1998b) makes the point that metadata is subject to change and needs to be tracked.

Traditionally, the data warehouse database is considered to be the end result of data warehouse processing. When the data finds its way into the database, it is available to be queried upon. This implies that the data travels in one direction from the operational systems into the data warehouse where it remains. There are increasing requirements for information from the data warehouse to be fed back into the operational systems. For instance, if it is possible for a data warehouse to report upon the profitability of individual customers, then the value of each customer is a useful piece of information to hold against the customer's record in the

operational system. This means that when, for instance, a customer contacts the organisation, the operator dealing with the call can be made immediately aware of the value that particular customer brings to the organisation and can behave accordingly. This is quite a new idea, which is hinted at by Sprukalis (1996), that could add a further spur to the development of data warehouses.

Data warehousing is seen by Stodder (1998) as an integral part of the future of what he calls 'intelligent enterprises'. It is my view that, in future, new operational applications will be developed with decision support requirements in mind. Part of the requirement gathering process for all new applications will encompass the contribution that the new application can make towards the information requirements of the organisation. The advent of temporal databases will make the development of data warehouses much easier from the time perspective. Simon (1998c) predicts that they will be commonplace by 2003 but, in my view, this seems somewhat optimistic. He also says that part of the reason that temporal databases have not been developed further is because data warehouses have satisfied the need to handle time by use of the time dimension. This, as I hope I have shown, is completely incorrect.

'What experience and history teach is this –

that nations and governments have never learned anything from history,

or acted upon any lessons they might have learned from it'. Hegel (1830)

References

Ackerman D (1991), Mystic Communion of Clocks, Jaguar of Sweet Laughter: New and selected poems, Random House Publishing, ISBN (0-679-74304-9)

Atre S (1998a), The art of data design, Computerworld, June 29 1998, V32 N26 P(71-72).

Atre S (1998b), Getting to know you data, Computerworld, August 24 1998, V32 N34 P(49-50).

Atre S (1998c), Beware dirty data, Computerworld, September 21 1998, V32 N38 P(67-69).

Bair J (1996), It's About Time! Supporting Temporal Data in a Warehouse, InfoDB Magazine V10 N1, February 1996

Batini C, Ceri S & Navathe S B (1992), Conceptual Database Design, an entity-relationship approach, Benjamin Cummings Publishing Company, ISBN (0-8053-0244-1)

Batini C, Talamo M & Tamassia R (1984), Computer aided layout of entity relationship diagrams, The Journal of Systems and Software, V4, P163(11),

Berson A & Smith S (1997), Data warehousing, data mining & OLAP, McGraw Hill, ISBN (0-07-006272-2).

Bohn K (1997a), Converting data for warehouses, DBMS, June 1997, V10 N7 P61.

Brackett M (1996), The Data Warehouse Challenge: Taming Data Chaos, John Wiley & Sons, ISBN(0-471-12744-2).

Bruce T A (1992), Designing Quality Databases with IDEF1X Information Models, Dorset House Publishing, ISBN (0-932633-18-8).

Calloway E (1995), The flavours of OLAP, PC Week, July 17 1995, V12 N28 P14(2).

Celko J & McDonald J (1995), Don't warehouse dirty data, Datamation, October 5 1995, V41 N19 P42(5).

Checkland P (1981), Systems thinking systems practice, John Wiley & Sones, ISBN (0-471-27911-0)

Checkland P & Scholes J (1990), Soft systems methodology in action, John Wiley & Sons, ISBN (0-471-92768-6)

Chen P P-S (1976), The Entity-Relationship Model - Toward a Unified View of Data, ACM Transactions on Database Systems, 1(1):9-36, March 1976.

Cippola E (1996), Data warehouse implementation considerations, Enterprise Systems Journal, June 1996, V11 N6 P22(4).

Clifford J, Dyreson C, Isakowitz T, Jensen C & Snodgrass R (1997), On the semantics of 'now' in databases, ACM Transactions on Database Systems, V22 N2, p171(34), June 1997

Codd E F, Codd S B & Salley C T (1993), Providing OLAP to User Analysts: An IT Mandate, White Paper, E.F Codd and Associates.

Coleridge S T (1835), Table Talk, Oxford Dictionary of Quotations (1996), Oxford University Press, ISBN(0-19-860058-5)

Demarest M (1998), Improving data legibility in decision support systems, DBMS, May 1998, V7 N5 P55(7)

DeMarco T (1979), Structured analysis and system specification, Prentice Hall, ISBN(0-13-854380-1)

Devlin B (1997), Data Warehouse: from Architecture to Implementation, Addison Wesley, ISBN (0-201-96425-2)

Dodge G & Gorman T (1998), Oracle 8 Data Warehousing, John Wiley & Sons, ISBN(0-471-19952-4)

Elkins S (1998a), OLAP's place in the warehouse architecture, DBMS, April 1998, V11 N4 p44(2).

Elkins S (1998b), Open OLAP, DBMS, April 1998, V11 N4 p34(6).

Elmasri R & Navathe S B (1994), Fundamentals of Database Systems, Benjamin Cummings Publishing Company, 2nd Edition.

Elmasri R & Kouramajian V (1993), A Temporal Query Language for a Conceptual Model, Lecture Notes in Computer Science, Advanced Database Systems, Berlin Spinger Verlag, pages 175-190.

Elmasri R, El-Assal I & V. Kouramajian V (1990a), Semantics of Temporal Data in an Extended ER Model, 9th International Conference on the Entity Relationship Approach, pages 239-254, Lausanne, Switzerland, October 1990.

Elmasri R & Wuu G T J (1990b), A Temporal Model and Query Language for ER Databases, Proceedings of the 6th International Conference on Data Engineering, pages 76-83, 1990.

Fadlalla A (1996), The advancement of data warehousing, Enterprise Systems Journal, October 1996, V11 N10 P18(4).

Fayyad U, Piatetsky-Shapiro G & Smythe P (1996), The KDD process for extracting useful knowledge from volumes of data, Communications of the ACM, November 1996, V39 N11 P(27-34).

Ferg S (1985), Modelling the Time Dimension in an Entity Relationship Diagram, 4th International Conference on the Entity Relationship Approach, pages 280-286, Silver Spring, MD, Computer Society Press.

Ferrera R & Naecker P (1993), The data warehouse: a giant step forward, DEC Professional, November 1993, V12 N11 P26(3).

Ferrera R (1994), Building the data warehouse, DEC Professional, October 1994, V13 N11 P31(7).

Foley J (1996), Towering terabytes, Information week, September 30 1996, N599 P34(6).

Foley J (1997), Data warehouse pitfalls, Information week, May 19 1997, N631 P93(3).

French C D (1995), One Size Fits All – Database Architectures that do not work for DSS, ACM SIGMOD.

Gardner S (1998), Building the data warehouse, Communications of the ACM, September 1998, V41 N9 P(52-60).

Gaudin S (1996), Building a better data warehouse, Digital News and Review, June 1996, V13 N6 P22(3).

Green H, Deacon K & Down D (1997), Housing in England 1996/7, ISBN (0-11-621020-6)

Griffin J (1997), Methodologies for the future, Software magazine, February 1997, V17 N2 PS5(3).

Hammer J, The Stanford Data Warehousing Project, Technical Paper , Stanford University Department of Computer Science.

Hammergen T (1998), Official Sybase: Data warehousing on the internet, Sybase Press, ISBN (1-85032-857-9)

Hegel G.W.F (1975), Lectures on the philosophy of world history, 1830, translated by H.B Nisbet, Oxford Dictionary of Quotations (1996), Oxford University Press, ISBN(0-19-860058-5)

Hurwitz J (1997), The evolution of metadata, , DBMS, July 1997, V10 N8 P12(2).

Inmon W H (1992); Building the data warehouse; Wiley-QED; ISBN (0-471-56960-7)

Inmon W H & Hackathorn R D (1994), Using the data warehouse, John Wiley & Sons, ISBN (0-471-05966-0)

Inmon W H, Zachman J A & Geiger J G (1997), Data stores, data warehousing and the Zachman framework, McGraw-Hill, ISBN (0-07-031429-2).

Inmon W H (1998), Referential integrity in the data warehouse, Enterprise Systems Journal, August 1998, V13 N8 P46(2).

Inmon W H, Welch J D & Glassey K L (1997), Managing the data warehouse, John Wiley & Sons, ISBN (0-471-16310-4).

Inmon W H, Imhoff C & Battas G (1996), Building the operational data store, John Wiley & Sons, ISBN (0-471-12822-8).

Jensen C S, Clifford J, Elmasri R, Gadia S K, Hayes P, Jajodia S, Dyreson C, Grandi F, Kafer W, Kline N, Lorentzos N, Mitsopoulos Y, Montanari A, Nonen D, Peressi E, Pernici B, Roddick J F, Sarda N L, Scalas M R, Segev A, Snodgrass R T, Soo M D, Tansel A, Tiberio P & Wiederhold G (1994); A Consensus glossary of temporal database concepts; ACM SIGMOD Record; Vol 3. No 1; March 1994

Kamfonas M (1998), Where versions meet dimensions, Database Programming and Design, V11 N9, September 1998.

Kelly S (1996), Data warehousing the route to mass customisation, John Wiley & Sons, ISBN(0-471-96328-3), 1996.

Kelly S (1997), Data warehousing in action, John Wiley & Sons, ISBN (0-471-96640-1) 1997.

Kight B (1996), The smart way to build a data warehouse, Datamation, October 1996, V42 N16 P91(3).

Kimball R (1996a), The Data Warehouse Toolkit, John Wiley & Sons, ISBN (0-471-15337-0).

Kimball R (1994), Getting the Data Out, DBMS Magazine, Miller Freeman Inc, July 1994.

Kimball R (1995a), Why Decision Support Fails and How To Fix it, ACM SIGMOD Record, v24,n3,September 1995.

Kimball R (1995b), Is ER Modelling Hazardous to DSS, DBMS Magazine, V8 N11, October 1995

Kimball R (1996b), Defining Drill Dialect, DBMS Magazine, Issue 11, April 1996

[Kimb06] Kimball R (1996c), Automating Data Extraction, DBMS Magazine, V2 N3, August 1996

Kimball R (1996d), The building blocks of data extraction, DBMS Magazine, V2 N2, July 1996

Kimball R (1996e), Clean up your act, DBMS Magazine, V2 N5, October 1996

Kimball R (1996f), Handling Monster Dimensions, DBMS Magazine, V2 N1, June 1996

Kimball R (1996g), Past Masters, DBMS Magazine, Issue 12, May 1996

Kimball R (1996h), Mind your language, DBMS Magazine, Issue 9, February 1996

Kimball R (1998a), The Data Warehouse Lifecycle Toolkit, John Wiley & Sons, ISBN (0-471-25547-5) 1998

Kimball R (1998b), Help for Hierachies, DBMS Magazine, V11 N10 P(12-16), Sept 1998

Kimball R (1998c), Help for Dimensional Modelling, DBMS, V11 N9 P(14-17), August 1998

R Kimball (1997a), A dimensional modelling manifesto, DBMS, August 1997, V10 N9 P58(5).

[Kimb18] Kimball R (1996i), Drilling down, up and across, DBMS, March 1996, V9 N3 P14(3).

Kimball R (1996j), Factless fact tables, DBMS, September 1996, V9 N10 P16(2).

Kimball R (1996k), Causal (not casual) dimensions, DBMS, November 1996, V9 N12 P16(2).

Kimball R (1997b), It's time for time, DBMS, July 1997, V10 N8 P16(3).

Kimball R (1998d), Bringing up supermarts, DBMS, January 1998, V11 N1 P(47-53).

Kimball R (1998e), The operational data warehouse, DBMS, January 1998, V11 N1 P(14-16).

Kimball R (1998f), Meta meta data data, DBMS, March 1998, V11 N3 P(18-20).

Kimball R (1998g), Surrogate Keys, DBMS, May 1998, V11 N5 P(14-16)

Klopprogge M R (1981), TERM: An Approach to Include the Time Dimension in the Entity Relationship Model, Proceedings of the Second International Conference on the Entity Relationship Approach, pages 477-512, Washington DC, October 1981.

Klopprogge M R & Lockemann M C (1983), Modelling Information Preserving Databases: Consequences of the Concept of Time, The 9th International Conference on Very Large Databases, pages 399 – 416, October 1983.

Labio W J, Quass D, Adelberg B, Physical Design for Data Warehouses, Technical Paper, Stanford University Department of Computer Science.

Lai V S, Kuilboer J-P & Guynes J L (1994), Temporal Databases: Model Design and Commercialisation Prospects, Data Base, 25(3):6-18.

Martin J (1985), Diagramming techniques for analysts and programmers, Prentice-Hall, ISBN (0-13-208794-4)

Mattison R (1996), Data Warehousing: Strategies, Technologies & Techniques, McGraw-Hill, ISBN (0-07-041034-8)

McBrien P, Seltveit A H & Wangler B (1992), An Entity Relationship Model Extended to Describe Historical Information, International Conference on Information Systems and Management of Data, pages 244-260, Bangalore, India, July 1992.

McClanahan D (1997), Data Modelling for OLAP, Data Based Advisor, March 1997, V15 N3 P66(5).

Menefee C (1998), Eight firms dominate data warehousing, Newsbytes news network, August 17 1998, P1.

Menninger D (1995a), Breaking all the rules,: an insiders guide to practical normalisation, Data Based Advisor, Jan 1995, V13 N1 P116(5).

Menninger D (1995b), Seeing stars, Data Based Advisor, July 1995, V13 N6 P106(3).

Meyer D & Cannon C (1998), Building a Better Data Warehouse, Prentice Hall PTR, ISBN (0-13-890757-9)

Miller G A (1956), The magical number seven, plus or minus two: some limits on our capacity for processing information, Psychological Review, April 1994, V101 N2, P343(10) (reprinted from 1956).

Morse S & Isaac D (1998), Parallel Systems in the Data Warehouse, Prentice Hall, ISBN (0-13-680604)

Mott R (1998), Knowledge payback, Informationweek, Sept 14 1998, N700 P298.

Packer A (1998), TPC benchmark secrets, Database Programming and Design, September 1998, V11 N9 P(30-37).

Palmer I R (1978), Practicalities in applying a formal methodology to data analysis, CACI Inc, Proceedings from BCS Conference 'Data analysis for information system design', 29th June 1978.

Pasahow E (1997), Data warehousing comes of age, Digital Systems Report, Winter 1997, V19 N4 P24(3).

Paulin J (1998), Dispelling the mixed workload myth, Database Programming and Design, October 1998, V11 N10 P(32-37).

Poe V, Klauer P & Brobst S (1997), Building a Data Warehouse for Decision Support – Second Edition, Prentice Hall – 0-13-769639-6.

Raden N (1996), Modelling a data warehouse, Information Week, Jan 29 1996 N564 P60(6).

Rigney T (1996), Nothing beats a good design, Software Magazine, October 1996, V16 N10 PS7.

Rudin K (1996), What's new in data warehousing, DBMS, August 1996, V9 N9 Ps4(5).

Sakaguchi T & Frolick M N (1996), A Review of the Data Warehousing Literature, University of Memphis.

Scheier R (1994), Build a better data warehouse with a robust data model, PC Week, December 5 1994, V11 N48 P24(1).

Sen A & Jacob V (1998), Industrial strength data warehousing, Communications of the ACM, September 1998, V41 N9 P(28-31)

Simon A (1998a), 90 Days to the Data Mart, John Wiley & Sons 0-471-25194-1.

Simon A (1998b), So long, farewell, Database Programming & Design; San Francisco; Oct 1998; Volume: 11 Issue: 10

Simon A (1998c), The next 5 years, Database Programming & Design; San Francisco; Oct 1998; Volume: 11 Issue: 10 P(26-31)

Simon A (1998d), Scoping the data warehouse, Database Programming & Design; San Francisco; July 1998; Volume: 11 Issue: 7 P(23-25)

Simon A (1998e), Data warehousing architecture: a fresh look, Database Programming and Design, January 1998, V11 N1 P(19-20).

Snodgrass R T (1997), Slowly Changing Dimensions, White Paper, November 1997

Snodgrass R T, Ahn I, Ariav G, Batory D, Clifford J, Dyreson C E, Elmasri R, Grandi F, Jensen C S, Kafer W, Kline N, Kulkarni K, Leung C, Lorentzos N, Roddick J F, Segev A, Soo M D & Sripada S M (1994), TSQL2 Language Specification, TSQL2 Language Design Committee, Department of Computer Science, University of Arizona, Sept 1994.

Snodgrass R (1998), Special Series on Temporal Database, Part 1: Of Duplicates and SeptupletsDatabase Programming & DesignVolume 11 Issue: 6; San Francisco; June 1998.

Sprague R & Carlson E (1982), Building Effective Decision Support Systems, Prentice Hall.

Sprukalis P (1996), Turning data into knowledge, Computing Canada, September 26 1996, V22 N20 P36.

Squire C (1995), Data Extraction and Transformation for the Data Warehouse, ACM SIGMOD.

Stedman C (1996), Can data be too up-to-date, Computerworld, September 23 1996, V30 N39 P53(2).

Stodder D (1998), Setting our new course, Database Programming and Design, September 1998, V11 N9 P5.

Strong D, Yang L & Yang R (1997), Data quality in context, Communications of the ACM, May 1997, V40 N5 P(103-110).

Surveyer J (1998), Decision support cycle unique, , Informationweek, September 14 1998, N700 P315.

Switzer A-M (1997), A warehouse map to success, Computing Canada, September 29 1997, V23 N20 P35.

Tansel , Temporal Databases – Design, Theory and Implementation, pages 213-229, Benjamin/Cummings, ISBN (0-8053-2413-5)

Tauzovich B (1991), Towards Temporal Extensions to the Entity-Relationship Model, The 10th International Conference on the Entity Relationship Approach, pages 163-179, San Mateo, California, October 1991.

Violino B (1998), Defining IT innovation, Informationweek, September 14 1998, N700 P(58-67).

Visser S (1998), Teach yourself DB2 Universal Database, Macmillan Computer Publishing, ISBN 0-672-31278-6

Wallace P (1994), Building a data warehouse, Infoworld, Feb 21 1994, V16 N8 P56(2).

Widom J (1995), Research Problems in Data Warehousing, 4th International Conference on Information and Knowledge Management, November 1995

Winter R (1998), Introducing the data warehouse challenge, Database Programming and Design, February 1998, V11 N2 P(19-21).

Youngworth P (1995), OLAP spells success for users and developers, Data Based Advisor, December 1995, V13 N11 P38(12)

Yourdon E (1993), Model driven systems development, Prentice Hall, ISBN (0-13-285818-0)

Zagelow G & Bontempo C (1998), The IBM data warehouse architcture, Communications of the ACM, V41 N9 P(38-48), Sept 1998,

Zimanyi E, Parent C & Spaccapietra S (1997), TERC+: A Temporal Conceptual Model, International Symposium on Digital Media Information Base, Nara, Japan, November 1997.

Appendix A - Practitioners guide

The purpose of this section of the thesis is to provide assistance to data warehouse practitioners in the delivery of data warehouse applications. As with previous sections of the document, the practitioner's guide is subdivided into two main sections, conceptual design and logical design. The conceptual design stage consists of the development of a data model using the Dot Modelling method described in chapter four. The logical model is developed as an extension of the Dot modelling method. The solutions presented in the chapter four, together with some of the solutions presented in the review section, are incorporated into the Dot Modelling methodology where appropriate.

In view of the fact that this section is intended for use by practitioners, it has been written with practitioners in mind. It is important that the users of this method get the best possible results from using it and so it is the entire method, not just the technical components, that are described. The language and the style of this section are aimed at the intended final audience.

Conceptual Model

One of the benefits of dimensional models is that they are easy to understand by both technical and non technical people. The non-technical business people are precisely the kinds of individuals we must talk to, and engage with, if we are to help organisations to implement their business strategies by the provision of relevant, accurate and timely information.

Whether the data pool, referred-to by Codd (1993), is finally implemented as a dimensional model or a non dimensional model is not the concern right now. What is crucial right now is that, conceptually at least, the business information imperatives are understood clearly and precisely by our customers and ourselves as consultants.

So the purpose of the Dot Modelling technique is to use a dimensional modelling approach as a consultancy aid to assist us in helping our customers to understand their business information needs.

The other main component of a data warehouse is history. Data warehouses are temporal applications and there is a requirement to report upon trends over time and snapshots at particular points in time, often in the form of time series analyses. There is a tendency in data warehouses to assume that the dimensional data remains static over time. This is an invalid assumption and is a major cause of inaccuracy of results returned from queries executed on data warehouses. For instance, in the United Kingdom, ten percent of people move house each year. This means that, when joining fact information to a customer dimension, facts from the past are increasingly likely to be associated with dimensional attributes whose values are different from what they were when the fact was originally recorded. It is important, therefore, to ensure that the facts are related to the correct dimensional attributes. This point is described by Kimball (1996a) in his work on what he calls 'slowly changing dimensions'. There are some shortcomings in Kimball's solution that are not discussed in detail in this document but which do not enable the proper representation of time insofar as dimensions are concerned. This issue is resolved by the implementation of 'retrospection'. Retrospection is the capability to look back into the past and is described in the logical modelling section of this document.

The Dot Modelling Methodology

The methodology, as outlined above, will now be described in detail.

Experience shows that this process can be conducted through the use of two, highly structured, workshops. The use joint application development (JAD) workshops is now a well accepted practice. As Scheier (1994) says, holding one or more of these workshop sessions is a good way of gathering user requirements.

Each of the two workshops can be completed in two days.

The Information Strategy Workshop

The objective of the Information Strategy workshop is for the business people, within the customer's organisation, to develop their own Dot Model that reflects their own perception of their business.

The session should last for approximately two days. The emphasis is on the word *'workshop'*. Every participant must expect to work and not to be *lectured-to*.

Participants - Practitioners

There should be, at least, two consultants present at the workshop sessions. The ideal combination is to have one highly organised workshop facilitator and one business consultant who understands the methodology and the customer's business.

It is very useful to have a third person who is able to act as a 'scribe' to record the proceedings of the sessions.

Quite a substantial proportion of the work is actually done in teams. It is quite useful to have an extra person who can assist in wandering from team to team checking that everyone understands what is required and that progress is being made.

The success or failure of these sessions often has very little to do with the technical content of the workshop but has everything to do with softer issues such as:

- The ambience - are they enjoying actually being there ?
- The relationships between the participants
- The 'friendliness' of the presenters
- The comfort of the participants

It is helpful to have people from HP who are sensitive to these issues and can detect and respond to body-language messages, suggest breaks at the right time etc.

Also, where the native language of the participants is different from the native language of the consultants, great care must be taken to ensure that the message, in both directions is clear. Ideally, at least one of the HP consultants should be fluent in the language of the participants. If that is not possible, then someone from the customer organisation must step in to act as the interpreter. If an interpreter is required to translate every statement by the presenters, then the workshop would generally take about fifty percent longer than normal.

Participants - Customer

The Information Strategy workshop requires a mixed attendance of business people and IT people. The ideal proportions are two-thirds from the business and one-third from IT. It is very important that the session is not dominated by IT staff.

Getting several senior people together for two whole days can be a virtually impossible task.

The only way to achieve this is ensure that:

- **You have senior executive commitment**
- **You book their diaries in advance**

Its no good waiting until the week before, you won't get them to attend

There must be some representation from the senior levels of management. As a test of seniority, we would describe a senior person as one who is measured on the performance of the business. This is someone whose personal income is, to some extent, dependant on the business being successful. In any case, they must be able to influence the business strategy and direction. The key to being successful in data warehousing is to focus on the business not the technology. As Poe et al (1997) put it, there must be a clear business imperative for constructing the data warehouse. It is not enough to merely state that information is strategic. This tendency to cite business requirements as fundamental to success has many followers. Cipolla (1996) says that any information system should exist solely to support the firm's business objectives and that it is critical that the business goals are clearly defined. According to Alan Simon (1998d), one of the major goals, initially, is to build and validate the business case. Switzer (1997) goes further, saying that data warehouse projects must have senior level

sponsorship and that warehouses are built to provide critical data and information to help set an organisation's overall strategic direction.

The Information Strategy Workshop Process

The steps involved in running the workshop are now described. Some of the subsequent sections should be regarded as being for guidance only. Different consultants have differing approaches to facilitating workshops and these variations in technique should be maintained. Here is one method that has been found to be successful in the past.

The tables in the room should be organised in a 'horseshoe' shape rather than a classroom style as this greatly encourages interaction.

Plenty of writing space should be available. I personally favour white boards with at least two flip charts to record important points.

Introduction

Estimated Time: 30 Minutes

Explain why the team is assembled and what the objectives are.

The objective is:
By the end of the information strategy workshop
we will have built a conceptual model of the information requirements needed to support the business direction of the organisation

Do not assume that everyone in the room will know who all the others are. I have yet to attend a workshop session where everyone knew each other.

Get everybody to introduce themselves. This is, obviously, a good ice-breaking method that has been used successfully for many years. I always ask them for:

- Their Name (pronunciation is very important if the language is foreign to you).

- Position in the organisation

- Brief description of their responsibilities

I also ask them to tell us one thing, about themselves, which is not work related. Maybe a hobby or something.

I also like to make a note about which of them are business people and which are IT people.

It is a good idea to introduce yourself first and for your colleagues to follow you. This takes the pressure off the participants.

Next come the ground rules. These are important and here are the ones I use. I think they are just common sense but you never know who you are dealing with so it is best to be explicit:

1. Silence means assent. If people do not voice objections to decisions then they are deemed to have accepted them. It is of no use to anyone if, during the wrap-up session on day two, someone says something like "Well I didn't agree with your basic ideas in the first place!"

2. The sessions start on time. It is incumbent on the facilitator to state clearly and precisely when the next session is due to start. It is the responsibility of all attendees to be there at that time. So it is *absolutely vital* that **the Consultants are never late to sessions**.

3. Mobile telephones are not allowed.

4. Personal criticisms are not allowed. Its OK for them to criticise the consultants, or the method, but not each other

5. Only one person is allowed to be speaking at a time.

Always give them the collective authority to change the rules or add others.

Business Goals

Estimated Time: 30 Minutes

This is the first real 'business' session. The following question should be posed:

Does your organisation have business goals ?

This is a directed question in the sense that it should be asked of the senior business people present. You need to pick up the body language responses quickly because some people may feel:

1. Embarrassed that they don't know what their business goals are.

2. Uncomfortable about sharing their business goals with outsiders.

Experience shows that the former is more common than the latter. Most people have a vague idea about business goals and would offer things like:

- Increase market share
- Increase customer loyalty
- Increase profitability

What we need to get to is to be able to articulate business goals that have the following properties:

Business goals are:

1. Measurable

2. Time Bounded

3. Customer Focused

The third of these is not an absolute requirement but most business goals do seem to relate to customers in some way. So a well articulated business goal might be:

To increase customer loyalty

by 5% per annum

for the next three years

If you can get one of these types of goal from your business people, that is good. If you can get two or three that's excellent.

The most likely result is that you won't get any. That's OK too. Its just that, if the organisation has business goals and is prepared to share them, then it is sensible to use them.

Thinking about Business Strategy

Estimated Time: 1 hour

This is the first workshop exercise. We split them into groups with three or four people in a group. In order to get the best out of this exercise, the composition of the groups is important. This is why it is useful to know who the business people are and who the IT people are.

We need one senior business person, if possible, in each group. The remaining business people should be spread as evenly as possible throughout the groups and the IT people likewise. Don't allow the IT people to become dominant in any one group. IT people have a tendency to become 'joined-at-the-hip' in these sessions if allowed.

One way of achieving an even spread of business and IT people is as follows:

During the introduction session earlier, the non-speaking facilitator should note down the names and job titles of each of the participants and can form the groups independently. The attendees can then be simply informed as to which group they have been assigned.

The objectives of the exercise at this stage are:

- Decide upon the business goals (assuming none were forthcoming previously) or use the business goals previously presented. Each team, therefore, is working toward its own set of goals, rather than the entire class having common goals. This is perfectly OK. Each of the business people in the room may have their own goals that will be different from the others. One or two goals is sufficient. Each goal must possess the three properties listed above.

- Think about business strategy to help them to achieve the goals. This is a prioritised set of initiatives that the organisation might need to implement. It is unlikely that a single item, by itself, would be enough to make the goal achievable. The companion slide set includes a slide with an example of a business goal and a strategy.

- What steps would they need to take in order to put the strategy into operation. The steps will be driven by the prioritisation of the strategic components.

Be careful to allow enough time for this. This process invariably leads to serious discussion in the groups as, almost, everyone has a view on the single thing their organisation needs to do in order to become more successful. These discussions are, in the main, a good thing and should be allowed to run their course. About one hour should be allowed for this part and a break should be included.

It is worth recommending that each person spends the first fifteen minutes thinking about the problem, so that each has a contribution to make before the group convenes.

Make sure that you have sufficient room for the group discussions to be conducted without affecting each other.

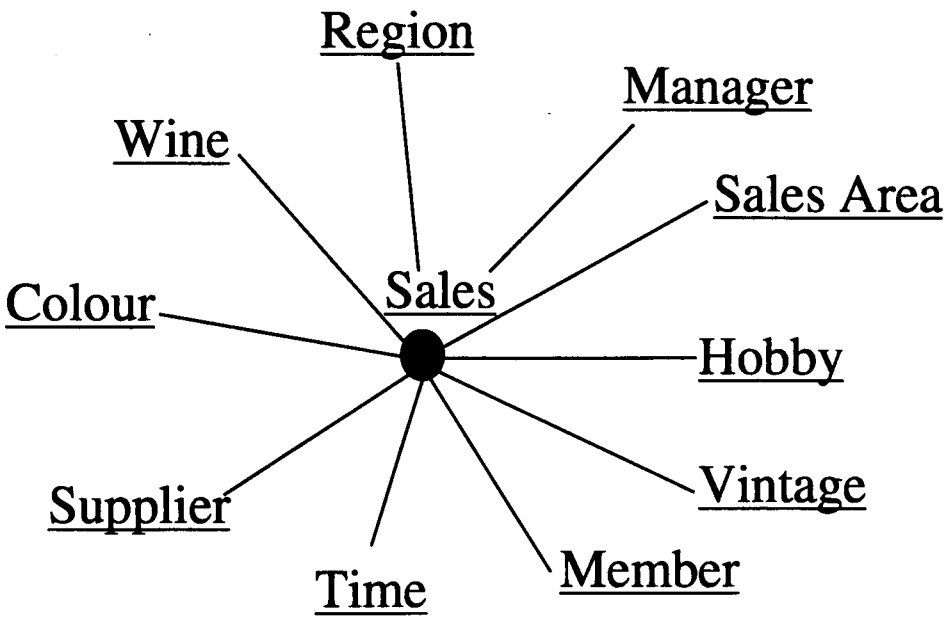
The groups must record their decisions as, later on, they will be expected to present them to the other groups.

The Initial Dot Model

Estimated Time: 1 - 2 hours

The class should re-convene for this teaching session. The objective is to get them to understand how to use the Dot Modelling system. This is quite easy but the IT people will understand it more quickly than the business people.

The best way to explain how to make a Dot Model is to start with a complete model, like the one below. It represents a mail order wine club.



Initial Dot Model for the Wine Club

The similarity to a star schema is obvious. It is a dimensional model just like a star schema or snowflake schema.

There are just three components to a Dot Model.

1. The Dot represents the measurable facts. Measurable facts are things like sales value, order quantity, call duration, call rated value. etc.

2. The dimensions, which may be organised into hierarchies (We don't introduce hierarchies yet).

3. Connectors, which are the lines that define the hierarchies.

It is vital that at least some of the members of the team understand how to do this. It is useful to ask questions, especially of the IT people, to convince yourself that each group is capable of developing a rudimentary model. At this stage, we do not expect perfect models and there is a refinement stage later on. However, it is important that they know the difference between a fact and a dimension. This is obvious to some people but experience shows it is not obvious to others. It helps to draw on the star schema analogy. Most IT people will be able to relate to it straight away. The business people will pick it up albeit more slowly.

Measurable Facts

Estimated Time: 15 minutes

We begin by explaining the meaning of the Dot, i.e. the measurable facts. These are the attributes of the Dot itself and they represent the focus of the model. Any data warehousing practitioner would be expected to be able to explain this. Typical facts are sales values and quantities. We must be careful to include derived facts such as profit or ROI. Each fact attribute must have the properties of being:

1. Numeric

2. Additive, or partially additive.

Each measurable fact must earn its place in the warehouse. Whoever suggests a fact should demonstrate that it has the capability to contribute towards the business strategy. Additionally, questions should be posed, in the form of queries (in natural language, not SQL), which will show how the fact will be used.

The measurable fact name, as it will be known in the data warehouse, should be recorded. It is vital that everyone has a clear idea as to the meaning of the facts and it makes sense to spend a little time focusing on the semantics of the information. In the 'Sales' scenario, the question 'What is a sale?' may result in a discussion about the semantics. For instance, an accountant may say that a sale has occurred when an invoice has been produced. A salesman might say

that a sale has occurred when an order is received. A warehouse manager might say that a sale has occurred when a delivery note is signed by the customer. Everyone has to be clear about the meaning of the data before moving on.

We also need to know the time granularity of the measurable fact. This should, ideally, be the lowest level of grain of time that is practical. The best possible level is known as the *'valid time'*. Valid Time means the time that the event occurred in real life. This usually translates to the transaction level. In a telecommunications application where the data warehouse is to record telephone calls, the valid time relates to the time the call was placed, right down to the second. In other applications, the time granularity might be set to 'day'.

Usually, all the facts will share the same time granularity. However, it is important to know where this is not the case so we record the time granularity for each measurable fact attribute.

For each fact attribute, metadata supporting the attribute should also be recorded. This means a precise description of the attribute in business terms.

The Dimensions

Estimated Time: 15 minutes

We now describe the meaning of dimensions. Again, it is expected that data warehousing practitioners understand this term and so the intention here is to give guidelines as to how to explain the term to non technical people.

One way of achieving this is to draw a two-dimensional spreadsheet with axes of, say, customer and product and explain that this is an example of a two dimensional view of sales. Each intersection, or cell, in the spreadsheet represents the sale of a particular product to a particular customer. I usually draw an intersection using lines and a dot where they intersect. The dot represents the measurable fact and, by using it, you reinforce the Dot notation in the minds of the participants. The two dimensional view can be extended to a three dimensional view by transforming the diagram into a cube and labelling the third axis as, say, 'time'. Now each intersection of the three dimensions, the dot, represents the sale of a particular product to a particular customer at a particular time. We have now reached the limit of what is possible

to draw. So we need a diagrammatic method that releases the Dot from its three dimensional limitation and this is where the Dot model helps. By describing the dimensions as axes of multidimensional spreadsheets, it is usually becomes a fairly easy to understand.

Creating the initial Dot Model

The task facing the groups is as follows:

Taking the business goals, the strategy and the steps they decided upon in the earlier group session, they now have to do the following:

1. Using their highest priority strategy, decide what information they need to help them. They will need guidance on this at first. Lead them through an example.

2. Formulate some questions that they would like to be able to ask.

3. Create a Dot Model that is able to answer those questions.

You need to allow at least one hour for this exercise and about half an hour for the teaching session. It is useful to include a major break, such as the lunch break, or overnight.

The facilitators must be on-hand all the time to assist where required. At this stage you will always get questions like "Is this a fact or a dimension ?". Also, the more 'switched-on' IT people will already start making assumptions about thing like:

- Availability of data from source systems

- Target database architectures

- Data quality issues etc.

They should be discouraged from these lines of thought. It is important that they do not constrain the innovative thinking processes. We are trying to establish the information we need, not what is currently available.

Group Presentations

Estimated Time: 30 minutes per group

Each group must now present, to the other groups:

1. Their business goals
2. The steps in their strategy
3. The information needed to support the steps
4. The Dot Model to support the information requirements
5. Some example questions that the model supports
6. How each question is relevant to their strategy

The information should be presented on a flip chart or overhead projector. The groups should each elect a spokesperson. Experience shows that some groups prefer to have two people, or even the whole group participating in the presentation.

The other groups should provide feedback. The facilitators MUST provide feedback. The feedback at this stage must always be positive with some suggestions as to how the model could be enhanced. Do not comment on the accuracy or completeness of the models. The refinement process will resolve these issues.

The Refinement Process

Estimated Time: 1 hour

We now have a classroom session about refinement of the model.

Firstly, we have to decide whether the groups should continue to work on their own models or whether all groups should adopt the same, perhaps a composite, model. There are no real preferences either way at this time. In either case, the work is still undertaken in groups.

Then we explain about:

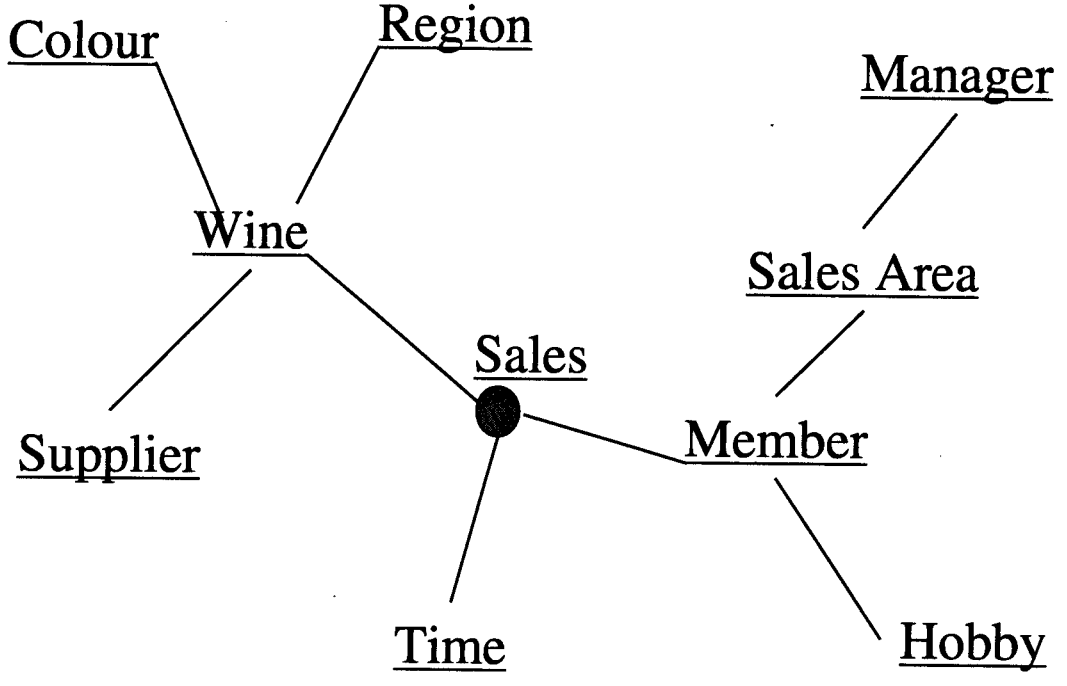
1. The combination of dimensions

2. Dimensional hierarchies

3. The inclusion of others' suggestions

Point one above concerns dimensions on the original model that are not really dimensions at all. There is an instance of this in the example provided in that the 'Vintage' dimension is really no more than an attribute of 'Wine'. After discussion, it is sometimes wise to leave these, apparently extraneous, dimensions alone if no agreement can be reached.

The creation of dimensional hierarchies is a fairly straightforward affair. The refined example Dot Model is as follows:



Refined Dot Model for the Wine Club

The groups should now be reformed to refine the models. The facilitators should wait for about ten to fifteen minutes and then approach the groups to see if they need help. There will

usually be some discussion and disagreement about how, and whether, changes should be incorporated.

Presenting the Refined Models

Estimated Time: 15 Minutes per Group

The refined models are re-presented to the other groups. The team has to show:

How the model has evolved from the original model as a result of the review process.

How any enhancements, from the previous feedback, have been incorporated.

Documenting the Models

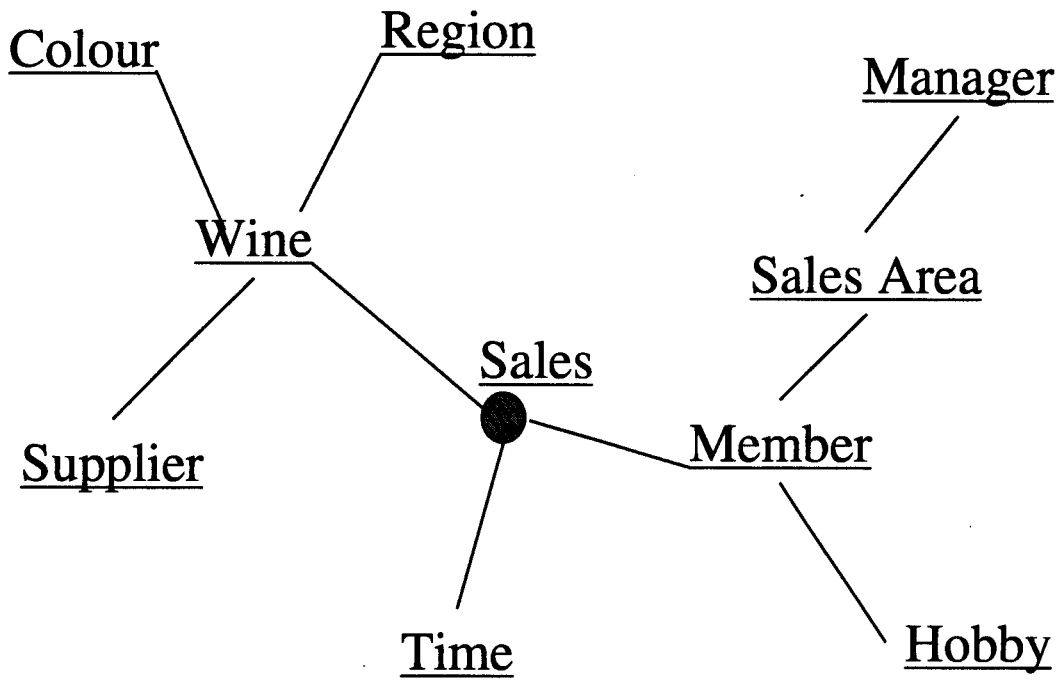
Estimated Time: 1 hour

The models are documented, in their refined state, using the ***'Data Model'*** and ***'Dimensions'*** worksheets. Encourage each group member to create their own. The facilitators must make sure they take a copy of each group's model.

These models provide the primary input to the Component Analysis workshop.

The data model and dimensions worksheets for the example are as follows:

Dot Modelling Data Model	
Model Name: Wine Club Sales	
Measurable Facts	
Fact Name	Metadata
Value	The total order line value (Quantity * Price) where the case price is used for quantities >= 12
Quantity	The number of bottles on the order line
Diagram	



Dot Modelling Dimensions		
Model Name	Wine Club Sales	
Dimension Name	**Retrospection**	**Frequency**
Hobby	Permanent	Monthly
Metadata The hobby dimension describes the member's main hobby		
Dimension Name	**Retrospection**	**Frequency**
Manager	True	Monthly
Metadata The manager dimension contains details of the managers of the sales areas.		
Dimension Name	**Retrospection**	**Frequency**
Member	True	Monthly
Metadata The member dimension contains details of the members of the club.		
Dimension Name	**Retrospection**	**Frequency**
Region	Permanent	Monthly
Metadata The regions relate to the wine growing areas such as Loire and Burgundy. In many cases, where districts are not applicable, the country name is used e.g. 'Chile'		
Dimension Name	**Retrospection**	**Frequency**
Sales Area	Permanent	N/A
Metadata The sales area dimension contains the name of the geographical areas in which the members reside.		
Dimension Name	**Retrospection**	**Frequency**
Supplier	False	Monthly
Metadata The Supplier dimension contains the name and details of the suppliers of wine.		
Dimension Name	**Retrospection**	**Frequency**
Wine	True	Monthly
Metadata The Wine dimension holds information about the wines sold by the club. Details include prices and costs		

The terms 'Retrospection' and 'Frequency' will be explained later.

Workshop wrap-up

Estimated Time: 30 Minutes

It is important to summarise the process that has been conducted over the previous two days and to congratulate the participants on having taken the first steps toward the creation of a 'Business-Led' information model for the organisation.

Explain how the model is just the first step and how it becomes the input to the next stage. Briefly describe the main components of the Component Analysis Workshop. Try to secure commitment from as many people as possible to attend the second workshop.

The arrangements for the second workshop should be confirmed, if that is possible.

Hand out the 'Attendee Feedback' form and ask each person to complete it. As this process is still very young, we need as much feedback as we can get.

While they are filling in the feedback forms, explain about dimensional attributes and try to encourage them to start thinking about this for the next workshop.

The Component Analysis Workshop

The principal objective of the Component Analysis Workshop is to put some substance into the model that was created in the Information Strategy Workshop.

Participants - Practitioners

As far as possible, the same Consultancy people, as were present in the first workshop, should participate in the second. This exercise is more technical than the first and so some additional skills are required but continuity of personnel should be preserved as far as possible.

Participants - Customer

Similarly, as far as is feasible, the same people, as before, should be present. However, as has been stated, this is a more technical workshop and some of the more senior business people should be permitted to send fully authorised deputies to work in their place. The business goals and the first part of the Dot Model have been established so their part of the task is complete.

The proportions of business people to IT people can be reversed in this workshop, so maybe two thirds IT people is OK this time. It is, however, vital that some business people are present.

What is also important is continuity, so I would reiterate the point that we need as many people as we can get from the first workshop to attend this second workshop.

The Component Analysis Workshop Process

The organisation of this second workshop is very similar to that of the first workshop. The room layout and writing facilities are the same. We do not need extra rooms for group work this time.

Review of Previous Model

Estimated Time: 30 Minutes

The purpose of this first exercise is to refresh the minds of the participants as to the state of the Dot Model as at the end of the first workshop.

Since the previous workshop, it is quite likely that the participants will have thought of other refinements to the model.

Whether these refinements should be adopted depends on the authority of the persons present. As long as any refinements have the full backing of the business, and can be demonstrated to be genuinely in pursuit of business goals, then they should be discussed and adopted where appropriate.

The extent of refinement is difficult to assess in general terms. Ideally, the lead consultant on the project will have kept in touch with developments inside the customer organisation and will be able to estimate the time needed to conduct this part of the exercise.

If there are no changes required, it should take no more than thirty minutes.

Dimensional Attributes

Estimated Time: 3-4 hours

This can be quite a time consuming process. The objective is to build a list of attributes for each dimension in the model.

Make sure lots of breaks are included in this session

Encourage people to walk around

There is a supporting Dot Modelling ***'Dimensional attributes'*** worksheet that should be used for this exercise. It is recommended that one of the facilitators maintains these worksheets as the session proceeds.

An example of this worksheet is as follows. Note that, although the form is shown as completed, some of the information is added later, in the logical modelling stage. Right now you should be aiming to record just the names of the attributes and the business metadata.

:

Dot Modelling Dimensional Attributes			
Model Name	Wine Club Sales		
Dimension Name	Member		
Attribute Name	Member Code		PK? Y
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The identifier for Members. This Code is supplied by the operational system.			
Source	**Transformation**	**Data type**	
Member Admin	None	Number 4	
Attribute Name	Member Name		PK? N
Retrospection	**Frequency**	**Dependency**	
False	Monthly	None	
Metadata: The name of the member in the form of surname preceded by initials			
Source	**Transformation**	**Data type**	
Member Admin	None	Char 25	
Attribute Name	Member Address		PK? N
Retrospection	**Frequency**	**Dependency**	
True	Monthly	Sales Area hierarchy	
Metadata: The member's address			
Source	**Transformation**	**Data type**	
Member Admin	None	Char 75	
Attribute Name	Date Joined		PK? N
Retrospection	**Frequency**	**Dependency**	
Permanent	N/A	None	
Metadata: This is the date the member joined the club.			
Source	**Transformation**	**Data type**	
Member Admin	Standard date conversion	Date	

One worksheet, or set of worksheets is completed for each dimension in the model. The identifying attribute should be included where known. The attribute name, as it will be known

in the data warehouse, should be recorded. The name must make it clear, to business people, what the attribute represents. For each attribute, metadata supporting the attribute should also be recorded. At this level we are looking for the following a concise but precise description of the attribute in business terms.

The facilitator should call for nominations of attributes from the workshop participants.

Each attribute must earn its place in the model. Whoever suggests an attribute should demonstrate that it has the capability to contribute towards the business strategy. Additionally, questions should be posed, in the form of queries (in natural language, not SQL) that will show how the attribute will be used.

As has been stated, this exercise can be very time consuming and, somewhat, tedious. It should be punctuated with frequent breaks.

Dimensional analysis of the facts

Estimated Time: 30 Minutes

Next, for each measurable fact attribute, we examine each dimension to determine the extent to which the standard arithmetic functions can be applied. This enables us to distinguish between fully additive facts and partially additive facts.

For instance:

- Supermarket quantities can be summed by product but not by customer.
- Bank balances can be summed at a particular point in time but not across time.
- Return on Investment, and any other percentages, cannot be added at all, but maximums, minimums and averages may be OK.
- So called 'Factless' facts, such as attendance can be counted but cannot be summed or averaged etc.

So each of the measurable facts should undergo an examination to determine which of the standard arithmetical functions can be safely applied to which dimensions.

The worksheet to be used to perform this exercise is called the ***'fact usage'*** worksheet and an example is as follows:

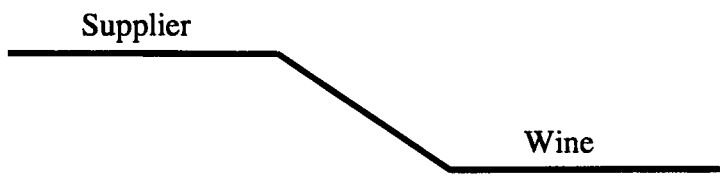
Dot Modelling Fact usage					
Model Name: Wine Club Sales					
Fact Name: Value		**Frequency** Daily			
Dimensions	Sum	Count	Ave	Min	Max
1) Manager	✓	✓	✓	✓	✓
2) Sales Area	✓	✓	✓	✓	✓
3) Member	✓	✓	✓	✓	✓
4) Hobby	✓	✓	✓	✓	✓
5) Colour	✓	✓	✓	✓	✓
6) Region	✓	✓	✓	✓	✓
7) Wine	✓	✓	✓	✓	✓
8) Supplier	✓	✓	✓	✓	✓
9) Time	✓	✓	✓	✓	✓
10)					

Hierarchies and Groupings

Estimated Time: 1 hour

This exercise is simply to provide some metadata to describe the relationships between the different levels in the Dimensional hierarchy. Similar to before, the metadata is required to provide a precise description of the relationship in business terms.

The worksheet used to capture this information is called the ***'Hierarchies and Groupings'*** worksheet and an example is as follows:

Dot Modelling Hierarchies & Groupings			
Model Name	Wine Club Sales		
			
Retrospection	True	**Frequency**	Monthly
Metadata **This hierarchy records the supply of wines to the club. Each wine may be supplied by one supplier at a time although, from time to time, the supplier for a particular wine may be changed.**			

The hierarchy or grouping is identified by recording the names of the dimensions in the Hierarchy diagram at the top of the worksheet. The example above shows Supplier → Wine grouping. The higher (Senior) level in the hierarchy is always placed at the top of the diagram and the lower (Junior) level is placed at the bottom.

Retrospection

Estimated Time: 1-2 hours

It will not have gone unnoticed that each of the dimensions, hierarchies and attributes must be given a value for ***Retrospection***.

Retrospection is the ability to look backwards into the past. Each of the database objects above can take one of three possible values for retrospection.

1. **True.**True retrospection means that the data warehouse must faithfully record and report upon the changing

2. **False**. False retrospection means that, while the object can change its value, only the latest value is required to be held in the data warehouse. Any previous values are permanently lost. This means that temporal support is not required.

3. **Permanent**. Permanent retrospection means that the values will not change during the life of the data warehouse. This means that temporal support is not required.

The meaning of retrospection varies slightly when applied to different types of warehouse object.

When applied to dimensions, the value for retrospection relates to the *'Existence'* of the dimension in question. For instance, If we need to know how many customers we have right now, then we must be able to distinguish between current customers and previous customers. If we want to be able to ask:

How many customers do we have today

compared to exactly one year ago ?

then we need to know exactly when a customer became a customer and when they ceased being a customer. Some customers may discontinuous lifespans. For some periods of time they may be active customers and during the intervening periods they may not be. If it is required to track this information faithfully, then *'Retrospection = True'* would apply to the Customer dimension.

Another example might be the 'Supplier' dimension. While we might be interested to know whether a supplier is a 'current' or 'dead' supplier, it is perhaps not so crucial to know when they became a supplier and when they ceased. So, in this case, *'Retrospection = False'* may well be more appropriate.

Still another example of dimensions is 'Region'. In some applications, the Regions might be expected to always exist. In these cases, there is no point in tracking the existence of the dimensions so we allow for a status of *'Retrospection = Permanent'* to accommodate this situation.

Insofar as Hierarchies and Attributes are concerned, retrospection relates to the values that are held by these objects.

If a customer moves from one region to another, or changes some important attribute such as, say, 'Number of Children', then, depending on the application, it may be important to be able to trace the history of these changes to ensure that queries return accurate results. If this is the case, then *'Retrospection = True'* should be applied to the attribute or hierarchy.

In other cases it may only be required to record the latest value, such as the customer's spouse's name. In this case, *'Retrospection = False'* would apply.

In still other cases the values may never change. An example of this is 'date of birth'. In these cases a value of *'Retrospection = Permanent'* would be more applicable.

The value given, to each data warehouse object, for Retrospection will become very important when the project moves into the design phase. It must be remembered that the requirements relating to retrospection are business, and not technical, requirements. It is all about accuracy of results of queries submitted by the users.

The measurable fact attributes,
once entered into the warehouse,
never change.
These attributes have an implied value of:
Retrospection = Permanent

Therefore, we have to examine each dimension, dimensional attribute and hierarchy to establish which value for retrospection should be applied. This means that the work sheets, previously completed, must be re-visited.

How to determine the value for retrospection

In order to establish the value for retrospection of each data warehouse object, the designer must investigate the use of the object within the organisation.

One way of doing this is to question the appropriate personnel as follows:

If this object were to change, would you want

to be able the track the history accurately ?

Experience has shown that the response to this type of question is, almost invariably, in the affirmative. As a result, every attribute becomes a candidate for special treatment with respect to time and every dimension, relationship and attribute will be assigned a value for retrospection of true. In view of the fact that provision of temporal support is expensive in terms of resources and adversely affects performance, it is very important that such support is only provided where it will truly add value to the information provided by the data warehouse. So there is a need for a practitioners methodology to adopt an approach to ascertaining the real need for temporal support on an object by object basis. The method should, as far as possible, provide an objective means of evaluation.

One approach is to ask questions that do not invite a simple 'yes' or 'no' response, as follows:

For dimensions:

How would the answer to the question: *How many customers do we have today compared to the same time last year ?* help you in making decisions regarding your business objectives. Other questions along similar lines such as: *How many customers have we had for longer than one year, or two years ?* should also be phrased and the answer should be expressed in terms that show how the ability to answer such questions would provide a clear advantage. This type of question can be answered only where retrospection is true for the dimension.

For relationships, the questions might be: *How many customers do we have in each sales area today compared to the same time last year ?* and for attributes: *How many of our customers have moved house in the last year ?*

The questions need to focus on the time requirements, preferably with a time series type of answer. The responder should be able to demonstrate how the answers to the questions would be helpful in the pursuit of their goals.

Granularity and the Dot_Time table

Estimated Time: 1-2 hours

We have discussed the time granularity of the measurable facts. Now we must discuss the time granularity of the dimensions, hierarchies and attributes. We also have to decide on the contents of the Dot_Time table.

For dimensions, hierarchies and attributes, the time granularity relates to the frequency with which changes to values, and changes to existence, are notified and applied to the data warehouse. So if, for example, customer updates are to be sent to the warehouse on a monthly basis, then the time granularity is monthly.

In order for complete accuracy to be assured, the granularity of time for the capture of changes in the dimensions must be the same as the granularity of time for the capture of the measurable fact events. In practice, this is hardly ever possible. There is an inevitable delay in time between the change event and the capture of that change in the source system. There is, usually, a further delay between the implementation of the change in the source system and the subsequent capture of the change in the data warehouse.

The challenge for data warehouse designers is to minimise the delays in the capture of changes.

The Dot Time worksheet is simply a list of all the headings, relating to time, by which the users will need to group the results from their queries. The requirements will vary from one application to another. Below is an example of the worksheet:

Dot Modelling - Time		
Model Name: Example		
Name	Description	Data Type
Day name	Standard day names	String
Day number	Day number 1-7 (Monday = 1)	Numeric
Bank holiday flag	Y = bank holiday	Character (Y/N)
Month number	Standard months 01-12	Numeric
Month name	Standard month names	String
Quarter	Standard quarters Q1, Q2, Q3, Q4	Numeric
Year	Year numbers	Numeric
Weekend day	Is it a Saturday or Sunday	Character (Y/N)
Fiscal month no.	April = 01, March = 12	Numeric
Fiscal quarter	Q1 = April - June	Numeric
24 hour opening	Y = open for 24 hours	Character (Y/N)
Weather conditions	Indicator for weather	Character

It is also important to establish how much history the application will wish to hold. This has two main benefits:

It tells the designers how many records will have to be created in order to populate the Dot Time table.

It gives the designers some indication of the ultimate size of the database when fully populated.

Also, check whether future data will be needed such as forecasts and budgets as these will have to be built into the Dot Time table as well.

Workshop wrap-up

Estimated Time: 30 Minutes

It is important to summarise the process that has been conducted over the previous two days. They now have a complete, business driven, conceptual information model that the designers can use to help to build the data warehouse.

Hand out the 'Attendee Feedback' form and ask each person to complete it. As this process is still very young, we need as much feedback as we can get.

The implementation of retrospection

During the conceptual modelling phase, the values for retrospection have been captured for every dimension, hierarchical relationship and attribute in the model. Each has been given a value of one of the following:

1. True

2. False

3. Permanent

The logical model has to support the requirements that are implicit in the three classifications above. In this section, guidance is provided on how to implement retrospection. The approach is slightly different for dimensions, relationships and attributes so separate, albeit similar, methods apply to each.

Dimensions

For a dimension, if retrospection is true, this means that it is the existence of the dimension that needs to be tracked with respect to time. A customer may have a long relationship with an organisation but their existence as a customer may be punctuated by periods of non existence. For instance, the Ford Motor Company may sell a new car to a customer. Two years later the customer may replace the car with a new Vauxhall and, after a further two years, may purchase another car from Ford. Over a period of five years, the Ford Motor Company would need to retain the customers details but should record the fact that, for a two year period, the customer could not really be counted as a customer.

This should be implemented by the use of an existence attribute in the form of a period which, in relational database terms, translates to a new relation containing the dimensional key, a start time and an end time. The granularity of the period will depend upon the requirements of the application. This kind of solution enables time series analyses to be performed. So questions like: *How many customers do we have now compared to the same time last year ?* are possible using this method. The update method is to add a new existence record each time the existence of a

customer changes. The latest record should have the end time left open to indicate that the customer still exists and it should be closed if the customer has ceased to exist.

If retrospection is false, for dimensions this means it is important to know whether or not the dimension exists. Using the customer example, it means that we need to know whether the customer really is a customer or not. It is not necessary to record the history of customer existence. This is best implemented by the use of a Boolean existence attribute that has true or false values and simply tells us whether the dimension exists or not. This approach enables questions like: *How many customers do we have right now ?* to be answered. The update method to be applied would be Kimball's type one approach which is to overwrite the old values. A novel approach to this is to call the existence attribute *'Active'* and for it to have a data type of numeric with a range of values of NULL, which represents non existence and '1' that represents existence. This enables the following queries to be executed:

Select count(Active) from customer

or

Select sum(Active) from customer

The way null values are treated will ensure that the above queries will return the correct answers.

Permanent retrospection with respect to dimensions requires no support at all as it means that the dimension is expected to exist for all time. Examples of dimensions with permanent retrospection tend to occur higher up the dimensional hierarchies. Geographically stable areas such as countries would usually fall into this category as they would be expected to remain in existence for all time insofar as the data warehouse is concerned.

Relationships

Within this context, a relationship describes some connection between two dimensions. In other words, a dimensional hierarchy. True retrospection means that it is necessary to track the value of the relationship over time. This means tracking the existence of instances of the

relationship. The method is the same irrespective of whether the relationship is explicit, in the form of a snowflake schema, or implicit as in a denormalised star.

The way to implement this depends on whether there is a need for time series analysis to be performed on the relationship. If, for instance, customers are segregated into geographical sales areas, then there exists a relationship between a customer dimension and a sales area dimension. If there is a need to analyse customers within sales areas over time, then this must be implemented with an existence attribute for the relationship. The existence attribute has to take the form of a period consisting of a start time and an end time. In relational terms, this means the creation of a new relation. If, however, there is no requirement for time series analysis of the relationship, then a further choice has to be made. If the dimension that is logically higher in the hierarchy has the property of true retrospection, then an intersection relation is required to implement the relationship. If, however, the higher dimension does not have true retrospection, then Kimball's type two method can safely be used without the threat of extraneous cascaded inserts.

If retrospection is false, then Kimball's type one approach, which is a simple overwrite, should be used.

Where retrospection is permanent, then no implementation solution is necessary as the relationship will not change.

Attributes

If an attribute possesses the property of true retrospection, there is a need to track the value of that attribute over time.

The implementation of true retrospection for attributes is best achieved in one of two ways. If there is a requirement to perform time series analysis for the attribute, then its separate values must be tied to an existence attribute in the form of a time period that comprises a start time and an end time. This should be modelled, in relational terms, as an additional relation having attributes of dimension key, start time, end time and attribute value. If there is no requirement to analyse the attribute on a time series basis, then a further choice has to be made. If the attribute is the property of a dimension that is at the lowest level in the hierarchy

(i.e. it is directly attached to the fact table), then Kimball's type two solution can be used, otherwise the existence attribute approach should be adopted unless the number of extraneous cascaded inserts is manageable.

If retrospection is false, then Kimball's type one approach, which is a simple overwrite, should be used.

Where retrospection is permanent, then no implementation solution is necessary as the attribute will not change.

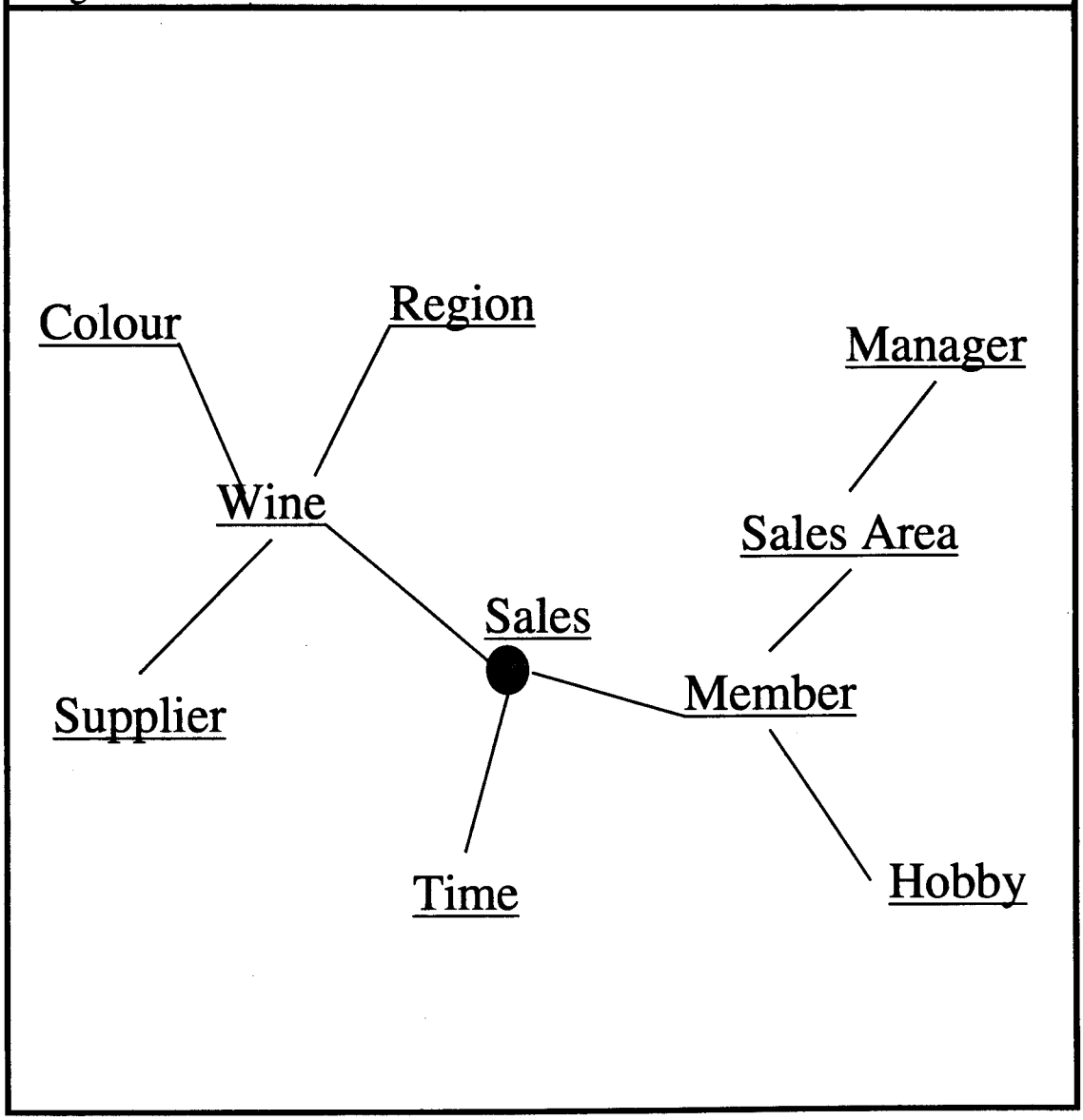
The following table captures the information above:

<table>
<tr><th></th><th colspan="2">Dimension</th><th colspan="3">Relationships</th><th colspan="3">Attributes</th></tr>
<tr><td rowspan="5">True</td><td colspan="2" rowspan="2">State duration or transition detection analysis required ?</td><td colspan="3" rowspan="2">State duration or transition detection analysis required ?</td><td colspan="3" rowspan="2">State duration or transition detection analysis required ?</td></tr>
<tr></tr>
<tr><td>Yes</td><td>No</td><td>Yes</td><td colspan="2">No</td><td>Yes</td><td colspan="2">No</td></tr>
<tr><td rowspan="3">Existence Period</td><td rowspan="3">Row Time stamp</td><td rowspan="3">Existence Period</td><td colspan="2">Does the Hierarchy Change Regularly?</td><td rowspan="3">Existence Period</td><td colspan="2">Is this attribute at the lowest level in the hierarchy?</td></tr>
<tr><td>Yes</td><td>No</td><td>Yes</td><td>No</td></tr>
<tr><td>Existence Period</td><td>Row Time stamp</td><td>Row Time stamp</td><td>Existence Period[1]</td></tr>
<tr><td>False</td><td colspan="2">Existence true/false attribute updated using type 1 method</td><td colspan="3">Type 1</td><td colspan="3">Type 1</td></tr>
<tr><td>Perm</td><td colspan="2">Will not change</td><td colspan="3">Will not change</td><td colspan="3">Will not change</td></tr>
</table>

[1] A row timestamp solution could be considered here if the resulting cascade inserts are of manageable proportions.

Appendix B – Complete Dot Model for the Wine Club

Dot Modelling Data Model	
Model Name: Wine Club Sales	
Measurable Facts	
Fact Name	Metadata
Value	The total order line value (Qantity * Price) where the case price is used for quantities >= 12
Quantity	The number of bottles on the order line
Diagram	



Dot Modelling Fact usage					
Model Name: Wine Club Sales					
Fact Name: Value		**Frequency** Daily			
Dimensions	Sum	Count	Ave	Min	Max
1) Manager	✓	✓	✓	✓	✓
2) Sales Area	✓	✓	✓	✓	✓
3) Member	✓	✓	✓	✓	✓
4) Hobby	✓	✓	✓	✓	✓
5) Colour	✓	✓	✓	✓	✓
6) Region	✓	✓	✓	✓	✓
7) Wine	✓	✓	✓	✓	✓
8) Supplier	✓	✓	✓	✓	✓
9) Time	✓	✓	✓	✓	✓
10)					

Dot Modelling Fact usage					
Model Name: Wine Club Sales					
Fact Name: Quantity		Frequency Daily			
Dimensions	Sum	Count	Ave	Min	Max
1) Manager	✓	✓	✓	✓	✓
2) Sales Area	✓	✓	✓	✓	✓
3) Member	✓	✓	✓	✓	✓
4) Hobby	✓	✓	✓	✓	✓
5) Colour	✓	✓	✓	✓	✓
6) Region	✓	✓	✓	✓	✓
7) Wine	✓	✓	✓	✓	✓
8) Supplier	✓	✓	✓	✓	✓
9) Time	✓	✓	✓	✓	✓
10)					

Dot Modelling Dimensions			
Model Name	Wine Club Sales		
Dimension Name		**Retrospection**	**Frequency**
Hobby		Permanent	Monthly
Metadata The hobby dimension describes the member's main hobby			
Dimension Name		**Retrospection**	**Frequency**
Manager		True	Monthly
Metadata The manager dimension contains details of the managers of the sales areas.			
Dimension Name		**Retrospection**	**Frequency**
Member		True	Monthly
Metadata The member dimension contains details of the members of the club.			
Dimension Name		**Retrospection**	**Frequency**
Region		Permanent	Monthly
Metadata The regions relate to the wine growing areas such as Loire and Burgundy. In many cases, where districts are not appliable, the country name is used e.g. 'Chile'			
Dimension Name		**Retrospection**	**Frequency**
Sales Area		Permanent	N/A
Metadata The sales area dimension contains the name of the geographical areas in which the members reside.			
Dimension Name		**Retrospection**	**Frequency**
Supplier		False	Monthly
Metadata The Supplier dimension contains the name and details of the suppliers of wine.			

Dimension Name	Retrospection	Frequency
Wine	True	Monthly
Metadata The Wine dimension holds information about the wines sold by the club. Details include prices and costs		

Dot Modelling Dimensional Attributes			
Model Name	Wine Club Sales		
Dimension Name	Hobby		
Attribute Name	Hobby Code		PK? Y
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The identifier for the Hobby. E.g. sa = 'Sailing'			
Source	**Transformation**	**Data type**	
K:\excel\work\hobbies.xls	None	Char 2	
Attribute Name	Hobby Name		PK? N
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The name of the hobby e.g 'Sailing'			
Source	**Transformation**	**Data type**	
K:\excel\work\sadesc.xls	None	Char 25	

Dot Modelling Dimensional Attributes			
Model Name	Wine Club Sales		
Dimension Name	Manager		
Attribute Name	Manager Code		PK? Y
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The identifier for managers.			
Source	**Transformation**	**Data type**	
Payroll	None	Char 2	
Attribute Name	Manager Name		PK? N
Retrospection	**Frequency**	**Dependency**	
False	Monthly	None	

Metadata: Manager's Name (First Name and Last Name)		
Source	**Transformation**	**Data type**
Payroll	None	Char 25

Dot Modelling Dimensional Attributes			
Model Name	Wine Club Sales		
Dimension Name	Member		
Attribute Name	Member Code		PK? Y
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The identifier for Members. This Code is supplied by the operational system.			
Source	**Transformation**	**Data type**	
Member Admin	None	Number 4	
Attribute Name	Member Name		PK? N
Retrospection	**Frequency**	**Dependency**	
False	Monthly	None	
Metadata: The name of the member in the form of surname preceded by initials			
Source	**Transformation**	**Data type**	
Member Admin	None	Char 25	
Attribute Name	Member Address		PK? N
Retrospection	**Frequency**	**Dependency**	
True	Monthly	Sales Area hierarchy	
Metadata: The member's address			
Source	**Transformation**	**Data type**	
Member Admin	None	Char 75	
Attribute Name	Date Joined		PK? N
Retrospection	**Frequency**	**Dependency**	
Permanent	N/A	None	
Metadata: This is the date the member joined the club.			
Source	**Transformation**	Data type	

Member Admin	Standard date conversion	**Date**

Dot Modelling Dimensional Attributes			
Model Name	Wine Club Sales		
Dimension Name	Region		
Attribute Name	Region Code		PK? Y
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The identifier for Region, eg: bur = Burgundy			
Source	**Transformation**	**Data type**	
K:\excel\work\catalog.xls	None	Char 3	
Attribute Name	Region Name		PK? N
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The name of the Wine growing region e.g.'Burgundy'			
Source	**Transformation**	**Data type**	
K:\excel\work\catalog.xls	None	Char 25	
Attribute Name	Country		PK? N
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The name of the country of origin of the wine. E.g. 'Spain'			
Source	**Transformation**	**Data type**	
K:\excel\work\catalog.xls			

Dot Modelling Dimensional Attributes			
Model Name	Wine Club Sales		
Dimension Name	Sales Area		
Attribute Name	Sales Area Code		PK? Y
Retrospection	**Frequency**	**Dependency**	
Permanent	N/A	None	
Metadata: The identifier for sales areas. E.g. 'NW'			
Source	**Transformation**	**Data type**	
K:\excel\work\sadesc.xls	None	Char 2	
Attribute Name	Sales Area Name		PK? N
Retrospection	**Frequency**	**Dependency**	
False	Monthly	None	
Metadata: The name of the geographical sales area e.g ' North West'			
Source	**Transformation**	**Data type**	
K:\excel\work\sadesc.xls	None	Char 25	

Dot Modelling Dimensional Attributes			
Model Name	Wine Club Sales		
Dimension Name	Supplier		
Attribute Name	Supplier Code		**PK? Y**
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The identifier for Supplier.			
Source	**Transformation**		**Data type**
Stock control system	None		Number 4
Attribute Name	Supplier Name		**PK? N**
Retrospection	**Frequency**	**Dependency**	
False	Monthly	None	
Metadata: The name of the Supplier			
Source	**Transformation**		**Data type**
Stock control system	None		Char 25
Attribute Name	Supplier Address		**PK? N**
Retrospection	**Frequency**	**Dependency**	
False	Monthly	**None**	
Metadata: The supplier's address			
Source	**Transformation**		**Data type**
Stock control system	None		Char 75
Attribute Name	Supplier Phone		PK? N
Retrospection	**Frequency**	**Dependency**	
False	Monthly	None	
Metadata: The supplier's phone number			
Source	**Transformation**		**Data type**
Stock control system	None		Char 25

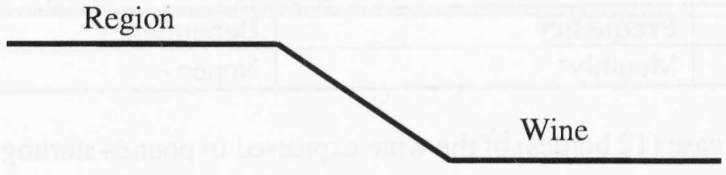
Dot Modelling Dimensional Attributes			
Model Name	Wine Club Sales		
Dimension Name	Wine		
Attribute Name	Wine Code		PK? Y
Retrospection	**Frequency**	**Dependency**	
Permanent	Monthly	None	
Metadata: The identifier for the wine. This code is supplied by the source system.			
Source	**Transformation**	**Data type**	
Stock Control System	None	Number 4	
Attribute Name	Wine Name		PK? N
Retrospection	**Frequency**	**Dependency**	
False	Monthly	None	
Metadata: The name of the wine as shown in the sales catalogue.			
Source	**Transformation**	**Data type**	
Stock Control System	None	Char 25	
Attribute Name	Vintage		PK?
Retrospection	**Frequency**	**Dependency**	
True	Monthly	None	
Metadata: The year that the wine was produced.			
Source	**Transformation**	**Data type**	
Stock Control System	Standard Date (Year)	Date (Year)	
Attribute Name	Alcohol by volume (ABV)		PK?
Retrospection	**Frequency**	Dependency	
False	Frequency	**None**	
Metadata: This is the alcoholic strength of the wine expressed as a percentage			
Source	**Transformation**	**Data type**	
Stock Control System	None	Number 2.1	
Attribute Name	Bottle Price		PK?
Retrospection	**Frequency**	**Dependency**	

True	Monthly	None
Metadata: The selling price of a bottle of the wine expressed in pounds sterling		

Source	**Transformation**	**Data type**
Stock Control System	None	Number 6.2

Dot Modelling Dimensional Attributes			
Model Name	Wine Club Sales		
Dimension Name	Wine (Continued)		
Attribute Name	Case Price		PK? N
Retrospection	**Frequency**	**Dependency**	
True	Monthly	None -	
Metadata: The selling price of a case (12 bottles) of the wine expressed in pounds sterling			
Source	**Transformation**		**Data type**
Stock control system	None		Char 6.2
Attribute Name	Bottle cost		PK? N
Retrospection	**Frequency**	**Dependency**	
True	Monthly	None	
Metadata: The cost to the Wine Club of a bottle of the wine expressed in pounds sterling			
Source	**Transformation**		**Data type**
Stock control system	None		Number 6.2

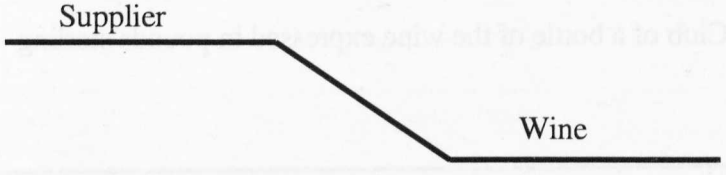
Dot Modelling Hierarchies & Groupings			
Model Name	WineClub Sales		



Retrospection	Permanent	**Frequency**	Monthly

Metadata

This grouping records the relationship between wines and the region in which the grapes are grown.



Retrospection	True	**Frequency**	Monthly

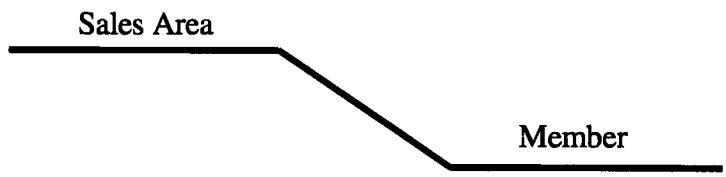
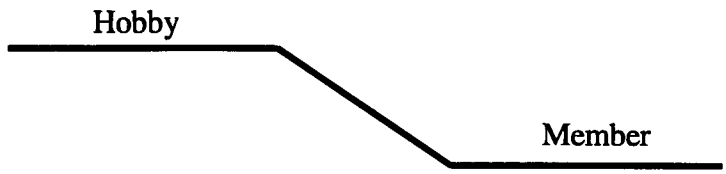
Metadata

This hierarchy records the supply of wines to the club. Each wine may be supplied by one supplier at a time although, from time to time, the supplier for a particular wine may be changed.

Retrospection	True	**Frequency**	Monthly

Metadata

This hierarchy records the management of a sales area. Each area has one manager at a time but, over time, managers may change.

Dot Modelling Hierarchies & Groupings			
Model Name	WineClub Sales		
			
Retrospection	True	**Frequency**	Monthly
Metadata This grouping records the relationship between members and the geographic sales areas in which they live. At any one time, a member belongs to one sales area but, over time, a member may move house and become resident in either the same or a different sales area.			
			
Retrospection	False	**Frequency**	Monthly
Metadata This grouping records the relationship between members and their principal pastime or hobby.			

Dot Modelling - Time		
Model Name:		
Name	Description	Data Type
Day	Standard day names	String
Month Number	Standard months 01-12	Numeric
Month Name	Standard month names	String
Quarter	Standard quarters Q1, Q2, Q3, Q4	Numeric
Year	Year numbers	Numeric
Weekend Day	Is it a Saturday or Sunday	Char (Y/N)
Fiscal Month no.	April = 01, March = 12	Numeric

Appendix C – The Wine Club database contents

This appendix lists the data stored in the Wine Club case study database. Many of the tables are fully listed but for some of the larger tables, samples have been extracted.

Member Dimension

Member Code	Member Name	Member Address	Sales Area Code	Hobby Code	Date Joined
1031	C.D. Anderson	77 Kingsway, Leeds, Yorks	NE	mr	30/12/94
1421	P.R. Anderson	21 Inglewood, Torquay, Devon	SW	cf	29/01/95
1136	A.G. Andrew	9 Broughton Hall Ave, Woking, Surrey	SE	sa	03/02/95
1233	C. Barford	15 Hadleigh Gardens, Carlisle, Cumbria	NW	sa	07/01/95
1651	E.D. Barnard	20 Orchard Grove, Blackwater, Surrey	SE	sa	07/01/95
1044	J.J. Booth	17 Church Hill, Ramsgate, Kent	SE	hr	28/12/94
1552	K.D. Cartwright	27 Sunnybank Rd, Leeds	NE	hr	13/01/95
1222	A.J. Carrick	8 Marlborough Court, Chester, Cheshire	NW	ff	08/03/95
1315	P. Chamberlain	11a Mount Pleasant, Sunderland	NE	cf	03/01/95
1231	D. Colville	13 Ballantyne Road, Minehead, Somerset	SW	sa	02/01/95
1543	M. Dowd	76 Wolseley Rd, Bristol	SW	sa	22/01/95
1313	C. Digby-Anderson	12 King George Cres, Leeds	NE	mr	27/12/94
1921	H. Drake	1a Lynch Rd, Manchester, Cheshire	NE	sa	15/03/95
1301	C. Eggar	79 Kings Ave, Leeds, Yorks	NE	ff	28/12/94
1241	P.D Elliot	21 Ingfield, Torquay, Devon	SW	cf	12/01/95
1316	A. Everett	92 Brighton Ave, Byfleet, Surrey	SE	hr	31/01/95
1323	C. Firkin	175 Harrow Heights, Carlisle, Cumbria	NW	ff	30/12/94
1561	D.F. Flanagan	19 Grove Road,	SE	sa	11/02/95

		Blackwater, Surrey			
1404	P.V. Frances	127 High Hill Rd, Ramsgate, Kent	SE	cf	31/12/94
1772	K.D. Gardiner	43 Park Rd, Leeds	NE	sa	09/01/95
1332	A.J. Gordon	82 Milton Ave, Chester, Cheshire	NW	sa	16/02/95
1135	S. Grant	53 Alder Road, Sunderland	NE	mr	08/01/95
1321	D. Hartley	88 Ballantyne Road, Minehead, Somerset	SW	hr	11/01/95
1453	K. Hitchcock	6 East Side Rd, Bristol	SW	hr	07/01/95
1133	C. Holroyd	76 Boxall Lane, Leeds	NE	sa	19/01/95
1291	H.Hughes	1 Bishops Sq, Manchester, Cheshire	NE	mr	29/12/94
1531	C.D. Jones	71 Queensway, Leeds, Yorks	NE	mr	03/01/95
4121	P.R. Jones	51 Tanglemere, Torquay, Devon	SW	cf	30/12/94
2236	G. Judge	10 Boxton Hall Road, Woking, Surrey	SE	sa	01/02/95
2133	B. Kerr	19 Purleigh Grove, Carlisle, Cumbria	NW	gf	02/01/95
6151	L. Kitson	22 Peartree Ave, Camberley, Surrey	SE	mr	02/03/95
1544	S.J. Knight	137 Leith Rd, Ramsgate, Kent	SE	mr	16/01/95
5152	L.A. Law	45 Moon Lane, Leeds	NE	gf	04/03/95
2122	S.G.T Lewis	21 Grange Road, Chester, Cheshire	NW	mr	21/01/95
3115	W. Lindsay	17 Midhope Close, Sunderland	NE	mr	21/01/95
2131	Q.E. McCallum	32 College Ride, Minehead, Somerset	SW	ff	10/01/95
5143	M. Mansell	54 St Margaret Rd, Bristol	SW	mr	30/12/94
3113	C. Maxwell	112 Longdene Rd, Leeds	NE	ff	11/01/95
9121	M.D.N. Newman	33 Walton Rd, Manchester, Cheshire	NE	sa	08/01/95
3101	C. Nibbs	2 Mounteagle, Leeds, Yorks	NE	mr	30/12/94
2141	J.K Noble	79 Priors Croft, Torquay, Devon	SW	gf	17/01/95
3116	T. Ogilvie	16 Martinswood, Byfleet, Surrey	SE	hr	27/12/94
3123	L.S. Olroyd	57 Guildown Rd, Carlisle,	NW	mr	10/01/95

		Cumbria			
5161	A.E. Ovenden	92 Cranmore Lane, Blackwater, Surrey	SE	sa	10/01/95
4104	I.F. Packman	24 Fortune Drive, Ramsgate, Kent	SE	cf	10/01/95
7172	T.J. Percival	24 Ashcroft, Leeds	NE	cf	04/01/95
3132	M. Phillips	29 Knowle Lane, Chester, Cheshire	NW	cf	08/01/95
6535	T.E Samuals	38 Rivers Close, Sunderland	NE	mr	13/02/95
3121	J. Sharp	3 Lodge Close, Minehead, Somerset	SW	cf	10/03/95
4153	C. Smallpiece	58 Ballard Road,Bristol	SW	cf	28/12/94
8733	A. Taylor	27 Rydens Way, Leeds	NE	hr	19/01/95
2191	R. Tripp	9 Waterden Road, Manchester, Cheshire	NE	cf	13/01/95

Wine Dimension

Wine code	Wine name	Col.	vintage	abv	region	bottle price	Case price	Supplier code	Bottle cost
2101	Claret Lot 278	r	1995	12.5	bor	3.49	39.79	1212	2.18
2201	Chateau Vieux	r	1990	12	bor	3.99	45.49	1212	2.49
2263	Chateau Heroult	r	1989	12	bor	3.59	40.93	1212	2.24
4371	Chateau la Perriere	r	1994	11	bor	4.49	51.19	1212	2.81
5436	Chateau Rouget	r	1995	12	bor	12.99	148.09	1212	8.12
3245	Bordeaux Blanc	w	1996	12	bor	2.99	34.09	1212	1.87
5987	Sauvignon Lot 279	w	1995	12	bor	3.49	39.79	1212	2.18
7658	Macon Rouge	r	1995	12.5	ber	3.99	45.49	4121	2.49
5558	Bougogne Pinot Noir	r	1995	12.5	ber	6.59	75.13	4121	4.12
1258	Nuits-St.George	r	1994	11.5	ber	9.99	113.89	4121	6.24
6764	Cotes de Beaune Villages	r	1985	12.5	ber	19.99	227.89	4121	12.49
4356	Macon Villages	w	1995	12.5	ber	3.99	45.49	4121	2.49
2317	Chablis Vieilles Vignes	w	1995	12.5	ber	8.99	102.49	4121	5.62
2187	Meursault 1 er Cru Chate	w	1994	12.5	ber	19.99	227.89	4121	12.49
2727	Loire Pinot Noir	r	1996	12	loi	3.99	45.49	3128	2.49
2765	Saumur Rouge	r	1996	12	loi	5.99	68.29	3128	3.74
5481	Sancerre Champigny Cuvee	r	1995	12	loi	9.99	113.89	3128	6.24
2885	Muscadet	w	1996	11	loi	2.99	34.09	3128	1.87
8865	Cheverny	w	1996	12	loi	5.99	68.29	3128	3.74
2766	Saumur Blanc	w	1996	12	loi	7.99	91.09	3128	4.99
3165	Vin de Pays des Bouches	r	1996	12	rho	2.99	34.09	4121	1.87
6746	Cotes du Rhone - Les Che	r	1996	12	rho	3.29	37.51	4121	2.06
2418	Cotes du Rhone Villages	r	1995	13	rho	4.39	50.05	4121	2.74
6745	Hermitage - Rouge - Gera	r	1991	12	rho	24.5	279.3	4121	15.31
3241	Rose De Anjou	p	1996	12	ros	3.49	39.79	2887	2.18
4433	Sancerre Rose	p	1995	12	ros	8.79	100.21	2887	5.49
5421	Valpolicella	r	1995	12	ita	3.69	42.07	2282	2.31
3421	Chianti Grati	r	1994	12.5	ita	3.99	45.49	2282	2.49
5654	Chianti Classico	r	1995	12	ita	5.49	62.59	2282	3.43

6621	Barolo	r	1990	12	ita	8.99	102.49	2282	5.62
7788	Barbaresco	r	1993	12	ita	9.99	113.89	2282	6.24
3457	Campora	r	1990	12	ita	25	285	2282	15.63
2221	La Mancha	r	1996	12	spa	2.19	24.97	3128	1.37
2621	Navajas Rioja	r	1995	12	spa	3.99	45.49	1543	2.49
2245	Marques de Grinon	r	1995	12	spa	4.49	51.19	3128	2.81
5643	Vina Pomal Rioja Crianza	r	1993	12	spa	4.99	56.89	1543	3.12
2222	La Mancha	w	1996	12	spa	2.19	24.97	3128	1.37
3399	Pedras do Monte	r	1995	12	por	3.29	37.51	3128	2.06
3119	Periquita	r	1994	12	por	4.29	48.91	3128	2.68
3210	Tinto de Anfora	r	1992	12	por	5.49	62.59	3128	3.43
4434	Sutter Home Vintner Sele	r	1995	12	usa	3.99	45.49	2887	2.49
4543	Sutter Home Cabernet Sau	r	1994	12	usa	4.69	53.47	2887	2.93
4678	Calera Pinot Noir	r	1993	11	usa	10.99	125.29	2887	6.87
4430	Sutter Home Vintner Sele	w	1995	12	usa	3.32	37.85	2887	2.08
9991	House Red	r	1997	12.5	hse	2.99	34	2887	2.14
9992	House White	w	1997	12.5	hse	2.99	34	3128	2.14

Region dimension

Region code	Region name	Country
bor	Bordeaux	france
ber	Bergundy	france
loi	Loire	france
rho	Rhone	france
hse	House Wine	Various
ros	Rose	france
ita	Italy	Italy
spa	Spain	Spain
usa	United States	United States
por	Portugal	Portugal

Manager dimension

Manager code	Manager name
M01	Gower
M02	Gooch
M03	Stewart
M04	Boycott

Sales area dimension

Sales Area code	Manager code	Sales area name
SE	M01	South East
NE	M02	North East
SW	M03	South West
NW	M04	North West

Supplier Dimension

Supplier code	Supplier name	Supplier address	Phone
1212	Vins Rouge (Wine Importers)	Unit 9, Castle View Industrial Park, Guildford, Surrey	01483 324768
1543	Vins XtraOrdinaire	Montague House, London Road, Sevenoaks, Kent	01732 658732
2282	Paige & Sons Ltd	22 South St, Maidstone, Kent	01622 567823
2887	Majestic Wine Warehouse	49 Mornington Place, Ramsgate, Kent	01304 223344
3128	HJR (Imports) PLC	27-31 Rochester Crescent, Southampton	01703 292444
3367	J Peter-Taylor	Egremont House, Shepherds Walk London NW3	0171 636 7224
4121	Cadet Piola	227 Quai de Orsay, Paris, France	
4166	Duhart Milon Rothschild	665-668 Avenue de Versailles, Paris, France	

Time dimension (sample)

Time code	Day name	Week end	Week	month	Month name	Season	Year
01/01/98	Thu	N	1	9801	January	Winter	1998
02/01/98	Fri	N	1	9801	January	Winter	1998
03/01/98	Sat	Y	1	9801	January	Winter	1998
04/01/98	Sun	Y	1	9801	January	Winter	1998
05/01/98	Mon	N	1	9801	January	Winter	1998
06/01/98	Tue	N	1	9801	January	Winter	1998
07/01/98	Wed	N	1	9801	January	Winter	1998
08/01/98	Thu	N	2	9801	January	Winter	1998
09/01/98	Fri	N	2	9801	January	Winter	1998
10/01/98	Sat	Y	2	9801	January	Winter	1998
11/01/98	Sun	Y	2	9801	January	Winter	1998
12/01/98	Mon	N	2	9801	January	Winter	1998
13/01/98	Tue	N	2	9801	January	Winter	1998
14/01/98	Wed	N	2	9801	January	Winter	1998
15/01/98	Thu	N	3	9801	January	Winter	1998
16/01/98	Fri	N	3	9801	January	Winter	1998

17/01/98	Sat	Y	3	9801	January	Winter	1998
18/01/98	Sun	Y	3	9801	January	Winter	1998
19/01/98	Mon	N	3	9801	January	Winter	1998
20/01/98	Tue	N	3	9801	January	Winter	1998
21/01/98	Wed	N	3	9801	January	Winter	1998
22/01/98	Thu	N	4	9801	January	Winter	1998
23/01/98	Fri	N	4	9801	January	Winter	1998
24/01/98	Sat	Y	4	9801	January	Winter	1998
25/01/98	Sun	Y	4	9801	January	Winter	1998
26/01/98	Mon	N	4	9801	January	Winter	1998
27/01/98	Tue	N	4	9801	January	Winter	1998
28/01/98	Wed	N	4	9801	January	Winter	1998
29/01/98	Thu	N	5	9801	January	Winter	1998
30/01/98	Fri	N	5	9801	January	Winter	1998
31/01/98	Sat	Y	5	9801	January	Winter	1998
01/02/98	Sun	Y	5	9802	February	Winter	1998
02/02/98	Mon	N	5	9802	February	Winter	1998
03/02/98	Tue	N	5	9802	February	Winter	1998
04/02/98	Wed	N	5	9802	February	Winter	1998
05/02/98	Thu	N	6	9802	February	Winter	1998
06/02/98	Fri	N	6	9802	February	Winter	1998
07/02/98	Sat	Y	6	9802	February	Winter	1998
08/02/98	Sun	Y	6	9802	February	Winter	1998
09/02/98	Mon	N	6	9802	February	Winter	1998
10/02/98	Tue	N	6	9802	February	Winter	1998
11/02/98	Wed	N	6	9802	February	Winter	1998
12/02/98	Thu	N	7	9802	February	Winter	1998
13/02/98	Fri	N	7	9802	February	Winter	1998
14/02/98	Sat	Y	7	9802	February	Winter	1998
15/02/98	Sun	Y	7	9802	February	Winter	1998
16/02/98	Mon	N	7	9802	February	Winter	1998
17/02/98	Tue	N	7	9802	February	Winter	1998
18/02/98	Wed	N	7	9802	February	Winter	1998
19/02/98	Thu	N	8	9802	February	Winter	1998
20/02/98	Fri	N	8	9802	February	Winter	1998
21/02/98	Sat	Y	8	9802	February	Winter	1998
22/02/98	Sun	Y	8	9802	February	Winter	1998
23/02/98	Mon	N	8	9802	February	Winter	1998
24/02/98	Tue	N	8	9802	February	Winter	1998
25/02/98	Wed	N	8	9802	February	Winter	1998
26/02/98	Thu	N	9	9802	February	Winter	1998
27/02/98	Fri	N	9	9802	February	Winter	1998
28/02/98	Sat	Y	9	9802	February	Winter	1998
01/03/98	Sun	Y	9	9803	March	Spring	1998

02/03/98	Mon	N	9	9803	March	Spring	1998
03/03/98	Tue	N	9	9803	March	Spring	1998
04/03/98	Wed	N	9	9803	March	Spring	1998
05/03/98	Thu	N	10	9803	March	Spring	1998
06/03/98	Fri	N	10	9803	March	Spring	1998
07/03/98	Sat	Y	10	9803	March	Spring	1998
08/03/98	Sun	Y	10	9803	March	Spring	1998
09/03/98	Mon	N	10	9803	March	Spring	1998
10/03/98	Tue	N	10	9803	March	Spring	1998
11/03/98	Wed	N	10	9803	March	Spring	1998
12/03/98	Thu	N	11	9803	March	Spring	1998
13/03/98	Fri	N	11	9803	March	Spring	1998
14/03/98	Sat	Y	11	9803	March	Spring	1998
15/03/98	Sun	Y	11	9803	March	Spring	1998
16/03/98	Mon	N	11	9803	March	Spring	1998
17/03/98	Tue	N	11	9803	March	Spring	1998
18/03/98	Wed	N	11	9803	March	Spring	1998
19/03/98	Thu	N	12	9803	March	Spring	1998
20/03/98	Fri	N	12	9803	March	Spring	1998
21/03/98	Sat	Y	12	9803	March	Spring	1998
22/03/98	Sun	Y	12	9803	March	Spring	1998
23/03/98	Mon	N	12	9803	March	Spring	1998
24/03/98	Tue	N	12	9803	March	Spring	1998

Hobby dimension

Hobby code	Hobby name
gf	Golf
cf	Coarse Fishing
ff	Fly Fishing
sa	Sailing
hr	Horse Racing
mu	Music
mr	Motor Racing

Sales Fact table (sample)

Member code	Wine code	Day	Quantity	Value
1031	2101	17/11/98	12	39.79
1421	2101	05/04/95	6	20.94
1136	2101	13/05/95	6	20.94
1233	2101	23/04/98	9	31.41
1651	2101	13/10/95	4	13.96
1044	2101	04/02/96	12	39.79
1552	2101	24/06/98	4	13.96
1222	2101	02/12/96	1	3.49
1315	2101	24/10/98	6	20.94
1231	2101	16/01/98	9	31.41
1543	2101	22/06/96	4	13.96
1313	2101	13/04/95	12	39.79
1921	2101	15/09/96	9	31.41
1301	2101	13/07/95	9	31.41
1241	2101	15/02/96	4	13.96
1316	2101	18/02/95	3	10.47
1323	2101	28/02/96	9	31.41
1561	2101	04/12/97	1	3.49
1404	2101	13/08/95	12	39.79
1772	2101	19/03/97	2	6.98
1332	2101	27/02/97	6	20.94
1135	2101	10/10/96	6	20.94
1321	2101	20/06/95	12	39.79
1453	2101	21/01/98	6	20.94

1133	2101	09/12/95	12	39.79
1291	2101	10/07/96	9	31.41
1531	2101	12/12/97	6	20.94
4121	2101	19/12/95	4	13.96
2236	2101	24/01/98	4	13.96
2133	2101	04/08/96	6	20.94
6151	2101	15/02/98	6	20.94
1544	2101	16/03/98	1	3.49
5152	2101	16/05/96	9	31.41
2122	2101	22/10/98	9	31.41
3115	2101	01/11/96	4	13.96
2131	2101	07/05/97	9	31.41
5143	2101	25/04/96	3	10.47
3113	2101	06/09/95	6	20.94
9121	2101	23/08/97	3	10.47
3101	2101	14/11/95	1	3.49
1772	9991	28/12/97	1	2.99
2141	2101	19/09/98	1	3.49
3116	2101	23/11/98	6	20.94
3123	2101	17/06/96	9	31.41
5161	2101	11/12/97	12	39.79
4104	2101	20/07/95	12	39.79
7172	2101	18/01/95	2	6.98
3132	2101	05/03/97	4	13.96
6535	2101	25/02/96	9	31.41
3121	2101	25/06/98	6	20.94
4153	2101	10/04/98	12	39.79
8733	2101	27/11/96	9	31.41
2191	2101	10/10/96	4	13.96
1031	2201	01/09/97	12	45.49
1421	2201	22/10/96	6	23.94
1136	2201	28/07/96	9	35.91
1233	2201	06/01/98	9	35.91
1651	2201	02/05/98	12	45.49
1044	2201	05/06/98	9	35.91
1552	2201	13/12/95	12	45.49
1222	2201	20/12/96	12	45.49
1315	2201	22/06/95	9	35.91
1231	2201	09/12/96	6	23.94
1543	2201	15/12/97	6	23.94
1313	2201	08/02/97	9	35.91
1921	2201	26/12/97	6	23.94
1301	2201	19/10/97	6	23.94
1241	2201	22/02/97	6	23.94

1316	2201	28/05/96	6	23.94
1323	2201	16/01/97	9	35.91
1561	2201	13/06/98	4	15.96
1404	2201	27/09/98	9	35.91
1772	2201	26/05/95	12	45.49
1332	2201	22/05/98	1	3.99
1135	2201	27/12/98	9	35.91
1321	2201	24/06/97	6	23.94
1453	2201	19/01/97	1	3.99
1133	2201	20/07/97	12	45.49
1291	2201	10/06/95	2	7.98
1531	2201	05/12/97	12	45.49
4121	2201	13/11/96	6	23.94
2236	2201	22/12/97	9	35.91
2133	2201	09/05/95	6	23.94
6151	2201	17/05/95	9	35.91
1544	2201	26/12/95	6	23.94
5152	2201	15/10/97	6	23.94
2122	2201	22/02/98	3	11.97