Chapter 3

Design Problems We Have to Face Up To

In this chapter, we will be reviewing the traditional approaches to designing data warehouses. During the review we will investigate whether or not these methods are still appropriate now that the business imperatives have been identified. We begin this chapter by picking up on the introduction to data warehousing.

DIMENSIONAL DATA MODELS

In Chapter 2, we introduced data warehousing and described, at a high level, how we might approach a design. The approach we adopted follows the style of some of the major luminaries in the development of data warehousing generally. This approach to design can be given the general description of dimensional. Star schemes and snowflake schemes are both examples of a dimensional data model.

The dimensional approach was adopted in our introduction for the following reasons:

- Dimensional data models are easy to understand. Therefore, they provide an ideal introduction to the subject.
- They are unambiguous.

- They reflect the way that business people perceive their businesses.
- Most RDBMS products now provide direct support for dimensional models.
- Research shows that almost all the literature supports the dimensional approach.

Unfortunately, although the dimensional model is generally acclaimed, there are alternative approaches and this has tended to result in "religious" wars (but no deaths as far as I know).

Even within the dimensional model camp there are differences of opinion. Some people believe that a perfect star should be used in all cases, while others prefer to see the hierarchies in the dimensions and would tend to opt for a snowflake design.

Where deep-rooted preferences exist, it is not the purpose of this book to try to make "road to Damascus" style conversions by making nonbelievers "see the light." Instead, it is intended to present some ideas and systematic arguments so that readers of this book can make their own architectural decisions based on a sound understanding of the facts of the matter. In any case, I believe there are far more serious design issues that we have to consider once we have overcome these apparent points of principle. Principles aside, we have also to consider any additional demands that customer relationship management might place on the data architecture.

A good objective for this chapter would be to devise a general high-level data architecture for data warehousing. In doing so, we'll discuss the following issues:

1. Dimensional versus third normal form (3NF) models
2. Stars versus snowflakes
3. What works for CRM

Dimensional Versus 3NF

There are two principal arguments in favor of dimensional models:

1. They are easy for business people to understand and use.
2. Retrieval performance is generally good.

The ease of understanding and ease of use bit is not in dispute. It is a fact that business people can understand and use dimensional models. Most business people can operate spreadsheets and a dimensional model can be likened to a multidimensional spreadsheet. We'll be exploring this in Chapter 5 when we start to investigate the dot modeling methodology.

The issue surrounding performance is just as clear cut. The main RDBMS vendors have all tweaked their query optimizers to enable them to recognize and execute dimensional queries more efficiently, and so performance is bound to be good in most instances. Even so, where the dimension tables are massively large, as the customer dimension can

be, joins between such tables and an even bigger fact table can be problematic. But this is not a problem that is peculiar to dimensional models. 3NF data structures are optimized for very quick insertion, update, and deletion of discrete data items. They are not optimized for massive extractions of data, and it is nonsensical to argue for a 3NF solution on the grounds of retrieval performance.

What Is Data Normalization?

Normalization is a process that aims to eliminate the unnecessary and uncontrolled duplication of data, often referred to as 'data redundancy'. A detailed examination of normalization is not within the scope of this book. However, a brief overview might be helpful (for more detail see Bruce, 1992, or Batini et al., 1992).

Normalization enables data structures to be made to conform to a set of well-defined rules. There are several levels of normalization and these are referred to as first normal form (1NF), second normal form (2NF), third normal form (3NF), and so on. There are exceptions, such as Boyce-Codd normal form (BCNF), but we won't be covering these. Also, we won't explore 4NF and 5NF as, for most purposes, an understanding of the levels up to 3NF is sufficient.

In relational theory there exists a rule called the e*ntity integrity rule*. This rule concerns the primary key of any given relation and assigns to the key the following two properties:

1. *Uniqueness.* This ensures that all the rows in a relational table can be uniquely identified.
2. *Minimality.* The key will consist of one or more attributes. The minimality property ensures that the length of the key is no longer than is necessary to ensure that the first property, uniqueness, is guaranteed.

Within any relation, there are dependencies between the key attributes and the non-key attributes. Take the following Order relation as an example:

Order

Order number	Primary key
Item number	Primary key
Order date	
Customer ID	
Product ID	
Product description	
Quantity	

Dependencies can be expressed in the form of "determinant" rules, as follows:

1. The Order Number determines the Customer ID.
2. The Order Number and Item Number determine the Product ID.
3. The Order Number and Item Number determine the Product Description.
4. The Order Number and Item Number determine the Quantity.
5. The Order Number determines the Order Date.

Notice that some of the items are functionally dependent on Order Number (part of the key), whereas others are functionally dependent on the combination of both the Order Number and the Order Item (the entire key). Where the dependency exists on the entire key, the dependency is said to be a *fully* functional dependency.

Where all the attributes have at least a functional dependency on the primary key, the relation is said to be in 1NF. This is the case in our example. Where all the attributes are engaged in a fully functional relationship with the primary key, the relation is said to be in 2NF. In order to change our relation to 2NF, we have to split some of the attributes into a separate relation as follows:

Order

Order number	Primary key
Order date	
Customer ID	

Order Item

Order number	Primary key
Item number	Primary key
Product ID	
Product description	
Quantity	

These relations are now in 2NF, since all the nonkey attributes are fully functionally dependent on their primary key.

There is one relationship that we have not picked up so far: The Product ID determines the Product Description. This is known as a *transitive* dependency because the Product ID itself can be determined by the combination of Order Number and Order Item (see dependency 2 above). In order for a relation to be classified as a 3NF relation, all transitive dependencies must be removed. So now we have three relations, all of which are in 3NF:

Order	
Order number	Primary key
Order date	
Customer ID	
Order Item	
Order number	Primary key
Item number	Primary key
Product ID	
Quantity	
Product	
Product ID	Primary key
Product description	

There is one major advantage in a 3NF solution and that is flexibility. Most operational systems are implemented somewhere between 2NF and 3NF, in that some tables will be in 2NF, whereas most will be in 3NF. This adherence to normalization tends to result in quite flexible data structures. We use the term *flexible* to describe a data structure that is quite easy to change should the need arise. The changing nature of the business requirements has already been described and, therefore, it must be advantageous to implement a data model that is adaptive in the sense that it can change as the business requirements evolve over time.

But what is the difference between dimensional and normalized?

Let's have another look at the simple star schema for the Wine Club, from Figure 2.3, which we first produced in our introduction in Chapter 2 (see Figure 3.1).

Some of the attributes of the customer dimension are:

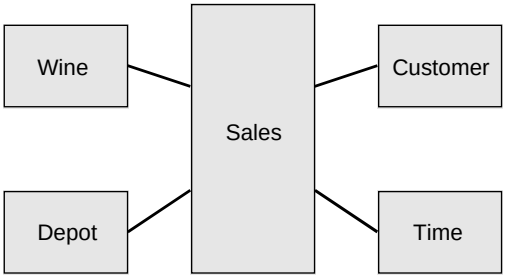

Figure 3.1 Star schema for the Wine Club.

Customer Dimension

Customer ID (Primary key)
Customer name
Street address
Town
County
Zip Code
Account manager ID
Account manager name

This dimension is currently in 2NF because, although all the nonprimary key columns are fully functionally dependent on the primary key, there is a transitive dependency in that the account manager name can also be determined from the account manager ID. So the 3NF version of the customer dimension above would look like this:

Customer Dimension

Customer ID (Primary key)
Customer name
Street address
Town
County
Zip Code
Account manager ID

Account Manager Dimension

Account Manager ID
Account Manager Name

The diagram is shown in Figure 3.2.

So what we have done is convert a star into the beginnings of a snowflake and, in doing so, have started putting the model into 3NF. If this process is carried out thoroughly with all the dimensions, then we should have a complete dimensional model in 3NF.

Well, not quite. We need also to look at the fact table. But first, let's go back to the original source of the Sales fact table, the Order, and Order Item table.

They have the following attributes:

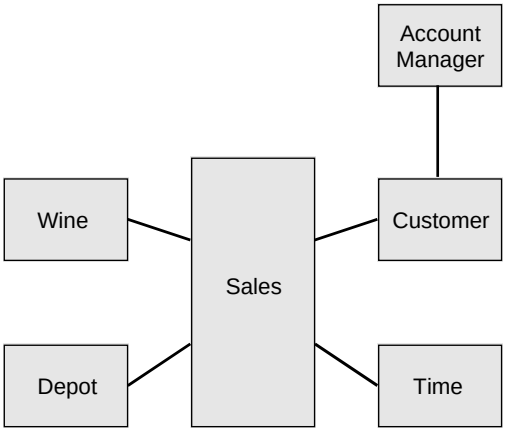

Figure 3.2 Third normal form version of the Wine Club dimensional model.

Order

Order number	(Primary key)
Customer ID	
Time	

Order Items

Order number	(Primary key)
Order item	(Primary key)
Wine ID	
Depot ID	
Quantity	
Value	

Both these tables are in 3NF. If we were to collapse them into one table, called Sales, by joining them on the Order Number, then the resulting table would look like this

Sales

Order number	(Primary key part 1)
Order item	(Primary key part 2)
Customer ID	
Wine ID	
Time	
Depot ID	
Quantity	
Value	

This table is now in 1NF because the Customer ID and Time, while being functionally dependent on the primary key, do not display the property of "full" functional dependency (i.e., they are not dependent on the Order Item).

In our dimensional model, we have decided not to include the Order Number and Order Item details. If we remove them, is there another candidate key for the resulting table? The answer is, it depends! Look at this version of the Sales table:

Sales	
Customer ID	(Primary key part 1)
Wine ID	(Primary key part 2)
Time	(Primary key part 3)
Depot ID	
Quantity	
Value	

The combination of Customer ID, Wine ID, and Time have emerged as a composite candidate key. Is this realistic? The answer is yes, it is. But, it all depends on the Time. The granularity of time has to be sufficiently fine as to ensure that the primary key has the property of uniqueness. Notice that we did not include the Depot ID as part of the primary key. The reason is that it does not add any further to the uniqueness of the key and, therefore, its inclusion would violate the other property of primary keys, that of minimality.

The purpose of this treatise is not to try to convince you that all dimensional models are automatically 3NF models; rather, my intention is to show that it is erroneous to say that the choice is between dimensional models and 3NF models. The following sums up the discussion:

1. Star schemes are not usually in 3NF.
2. Snowflake schemes can be in 3NF.

Stars and Snowflakes

The second religious war is being fought inside the dimensional model camp. This is the argument about star schema versus snowflake schema.

Kimball (1996) 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 that users might be intimidated by complex hierarchies.

My experience has shown his assertion, that users and business people are uncomfortable with 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 (1996) uses the hierarchy in Figure 3.3 as his example.

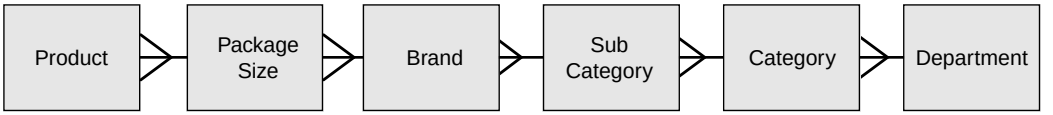

Figure 3.3 Confusing and intimidating hierarchy.

It is possible that it is this six-layer example in Figure 3.3 that is confusing and intimidating, rather than the principle. In practice, hierarchies involving so many layers are almost unheard of. Far more common are hierarchies like the one reproduced from the Wine Club case study, shown in Figure 3.4.

These hierarchies are very natural to managers because, in real-life scenarios, customers are organized into geographic areas 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 organized 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. In any case, if we present the users of our data warehouse with a star schema, all we have done, in most instances, is precisely that: We have flattened the hierarchy in a kind of "denormalization" process.

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.

Where does this get us? Well, in the dimensional versus 3NF debate, I suspect a certain amount of reading-between-the-lines interpretation is necessary and that what the 3NF camp is really shooting for is the retention of the online transaction processing (OLTP) schema in preference to a dimensional schema. The reason for this is that the OLTP model will more accurately reflect the underlying business processes and is, in theory at least, more flexible and adaptable to change. While this sounds like a great idea, the introduction of history usually makes this impossible to do. This is part of a major subject that we'll be covering in detail in Chapter 4.

It is worth noting that *all* online analytical processing (OLAP) products also implement a dimensional data model. Therefore, the terms *OLAP* and *dimensional* are synonymous.

This brings us neatly onto the subject of data marts. The question "When is it a data warehouse and when is it a data mart"? has also been the subject of much controversy.

Figure 3.4 Common organizational hierarchy.

Often, it is the commercial interests of the software vendors that carry the greatest influence. The view is held, by some, that the data warehouse is the big, perhaps enterprise-wide, repository and that data marts are much smaller, maybe departmental, extractions from the warehouse that the users get to analyze.

By the way, this is very closely associated with the previous discussion on 3NF versus dimensional models. Even the most enthusiastic supporter of the 3NF/OLTP approach is prepared to recognize the value that dimensional models bring to the party when it comes to OLAP. In a data warehouse that has an OLAP component, it is that OLAP component that the users actually get to use. Sometimes it is the only part of the warehouse that they have direct access to. This means that the bit of the warehouse that the users actually use is dimensional, irrespective of the underlying data model. In a data warehouse that implements a dimensional model, the bit that the users actually use is, obviously, also dimensional. Therefore, everyone appears to agree that the part that the users have access to should be dimensional. So it appears that the only thing that separates the two camps is some part of the data warehouse that the users do not have access to.

Returning to the issue about data marts, we decline to offer a definition on the basis that a subset of a data warehouse is still a data warehouse. There is a discussion on OLAP products generally in Chapter 10.

WHAT WORKS FOR CRM

Data warehousing is now a mature business solution. However, the evolution of business requires the evolution of data warehouses. That business people have to grasp the CRM nettle is an absolute fact. In order to do this, information is the key. It is fair to say that an organization cannot be successful at CRM without high-quality, timely, and accurate information. You cannot determine the value of a customer without information. You cannot personalize the message to your customer without information, and you cannot assess the risk of losing a customer without information.

If you want to obtain such information, then you really do need a data warehouse. In order to adopt a personalized marketing approach, we have to know as much as we can about our customers' circumstances and behavior. We described the difference between circumstances and behavior in the section on market segmentation in Chapter 1. The capability to accurately segment our customers is one the important properties of a data warehouse that is designed to support a CRM strategy. Therefore, the distinction between circumstances and behavior, two very different types of data, is crucial in the design of the data warehouse. Let's look at the components of a "traditional" data warehouse to try and determine how the two different types of data are treated.

The diagram in Figure 3.5 is our now familiar Wine Club example.

It remains in its star schema form for the purposes of this examination, but we could just as easily be reviewing a snowflake model.

The first two questions we have to ask are whether or not it contains information about:

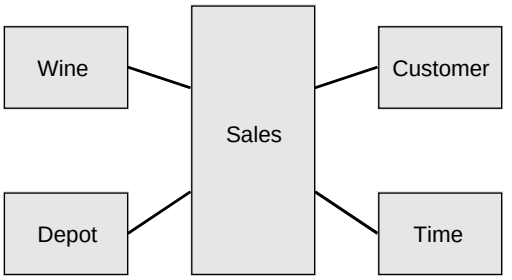

Figure 3.5 Star schema for the Wine Club.

1. Customers' behavior
2. Customers' circumstances

Clearly it does. The Sales table (the Fact table) contains details of sales made to customers. This is behavioral information, and it is a characteristic of dimensional data warehouses and data marts that the Fact table contains behavioral information. Sales is a good example, probably the most common, but there are plenty more from all industries:

- Telephone call usage
- Shipments and deliveries
- Insurance premiums and claims
- Hotel stays
- Aircraft flight bookings

These are all examples of the subject of a dimensional model, and they are all behavioral.

The customer dimension is the only place where we keep information about customer circumstances. According to Ralph Kimball (1996), the principal purpose of the customer dimension, as with all dimensions in a dimensional model, is to enable constraints to be placed on queries that are run against the fact table. The dimensions merely provide a convenient way of grouping the facts and appear as row headers in the user's result set. We need to be able to slice and dice the Fact table data "any which way." A solution based on the dimensional model is absolutely ideal for this purpose. It is simply made for slicing and dicing.

Returning to the terms *behavior* and *circumstances*, a dimensional model can be described as *behavior centric.* It is behavior centric because its principal purpose is to enable the easy and comprehensive analysis of behavioral data.

It is possible to make a physical link between Fact tables by the use of a common dimension tables as the diagram in Figure 3.6 shows.

This "daisy chain" effect enables us to "drill across" from one star schema to another. This common dimension is sometimes referred to as a *conformed* dimension.

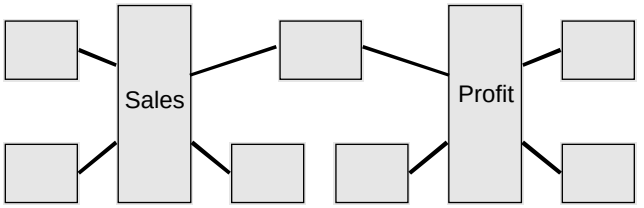

Figure 3.6 Sharing information.

We have seen previously how the first-generation data warehouses tended to focus on the analysis of behavioral information. Well, the second generation needs to support big business issues such as CRM and, in order to do this effectively, we have to be able to focus not only on behavior, but circumstances as well.

Customer Behavior and Customer Circumstances–The Cause-and-Effect Principle

We have explored the difference between customers' circumstances and their behavior, but why is it important?

Most of the time in data warehousing, we have been analyzing behavior. The Fact table in a traditional dimensional schema usually contains information about a customer's interaction with our business. That is: the way they behave toward us. In the Wine Club example we have been using, the Fact table contained information about sales. This, as has been shown, is the normal approach toward the development of data warehouses.

Now let us look again at one of the most pressing business problems, that of customer loyalty and its direct consequence, that of customer churn. For the moment let us put ourselves in the place of a customer of a cellular phone company and think of some reasons why we, as a customer, may decide that we no longer wish to remain as a customer of this company:

- Perhaps we have recently moved to a different area. Maybe the new area has a poor reception for this particular company.
- We might have moved to a new employer and have been given a mobile phone as part of the deal, making the old one surplus to requirements.
- We could have a child just starting out at college. The costs involved might require economies to be made elsewhere, and the mobile phone could be the luxury we can do without.

Each of the above situations could be the *cause* for us as customers to appear in next month's churn statistics for this cellular phone company. It would be really neat if the phone company could have predicted that we are a high-risk customer. The only way to do that is to analyze the information that we have gathered and apply some kind of predictive model to the data that yields a score between, say, 1 for a very low risk customer to 10 for a very high risk customer.

But what type of information is likely to give us the best indication of a customer's propensity to churn? Remember that, traditionally, data warehouses tend to be organized around behavioral systems. In a mobile telephone company, the most commonly used behavioral information is the call usage. Call usage provides information about:

- Types of calls made (local, long distance, collect, etc.)
- Durations of calls
- Amount charged for the call
- Time of day
- Call destinations
- Call distances

If we analyze the behavior of customers in these situations, what do you think we will find? I think we can safely predict that, just before the customer churned, they stopped making telephone calls! The abrupt change in behavior is the effect of the change in circumstances.

The cause-and-effect principles can be applied quite elegantly to the serious problem of customer churn and, therefore, customer loyalty. What we are seeing when we analyze behavior is *the effect of some change in the customer's circumstances*. The change in circumstances, either directly or indirectly, is the cause of their churning. If we analyze their behavior, it is simply going to tell us something that we already know and is blindingly obvious–the customer stopped using the phone. By this time it is usually far too late to do anything about it.

In view of the fact that most dimensional data warehouses measure behavior, it seems reasonable to conclude that such models may not be much help in predicting those customers that we are at risk of losing. We need to turn our attention to being very much more rigorous in our approach to tracking changes in circumstances, rather than behavior. Thus, the second-generation data warehouses that are being built as an aid to the development of CRM applications need to be able to model more than just behavior.

So instead of being behavior centric, perhaps they should be dimension centric or even circumstances centric. The preferred term is *customer centric*. Our second-generation data warehouses will be classified as customer centric.

Does this mean that we abandon behavioral information? Absolutely not! It's just that we need to switch the emphasis so that some types of information that are absolutely critical to a successful CRM strategy are more accessible.

So what does this mean for the great star schema debate? Well, all dimensional schemes are, in principle, behavioral in nature. In order to develop a customer-centric model we have to use a different approach.

If we are to build a customer-centric model, then it make sense to start with a model of the customer. We know that we have two major information types–behavior and circumstances. For the moment, let's focus in on the circumstances. Some of the kinds of things we might like to record about customers are:

Customer
Name
Address
Telephone number
Date of birth
Sex
Marital status

Of course, there are many, many more pieces of information that we could hold (check out Appendix D to see quite a comprehensive selection), but this little list is sufficient for the sake of example. At first sight, we might decide that we need a customer dimension as shown in Figure 3.7.

The customer dimension in Figure 3.7 would have some kind of customer identifier and a set of attributes like those listed in the table above. But that won't give us what we want. In order to enable our users to implement a data warehouse that supports CRM, one of the things they must be able to do is analyze, measure, and classify the effect of changes in a customer's circumstances. As far as we, that means the data architects, are concerned, a change in circumstances simply means a change in the value of some attribute. But, ignoring error corrections, not all attributes are subject to change as part of the ordinary course of business. Some attributes change and some don't. Even if an attribute does change, it does not necessarily mean that the change is of any real interest to our business. There is a business issue to be resolved here.

We can illustrate these points if we look a little more closely at the simple list of attributes above. Ignoring error corrections, which are the attributes that can change? Well, in theory at least, with the exception of the date of birth, they can all change. Now, there are two types of change that we are interested in:

1. Changes where we need to be able to see the previous values of the attribute, as well as the new value
2. Changes where the previous values of the attribute can be lost

What we have to do is group the attributes into these two different types. So we end up with a model with two entities like the one in Figure 3.8.

Figure 3.7 General model for customer details.

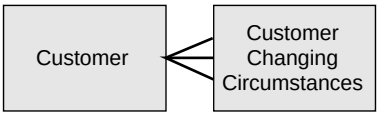

Figure 3.8 General model for a customer with changing circumstances.

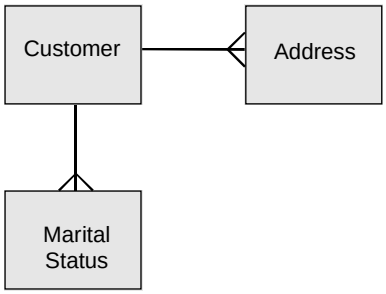

Figure 3.9 Example model showing customer with changing circumstances.

We are starting to build a general conceptual model for customers. For each customer, we have a set of attributes that can change as well as a set of attributes for which either they cannot change or, if they do, we do not need to know the previous values. Notice that the relationship has a cardinality of one to many. Please note this is not meant to show that there are many attributes that can change; it actually means that each attribute can change many times. For instance, a customer's address can change quite frequently over time.

In the Wine Club, the name, telephone number, date of birth, and sex are customer attributes where the business feels that either the attributes cannot change or the old values can be lost. This means that the address and marital status are attributes where the previous values should be preserved. So, using the example, the model should look as shown in Figure 3.9.

So each customer can have many changes of address and marital status, over time.

Now, the other main type of data that we need to capture about customers is their behavior. As we have discussed previously, the behavioral information comes from the customers' interaction with our organization. The conceptual general model that we are trying to develop must include behavioral information. It is shown in Figure 3.10.

Again the relationship between customers and their behavior is intended to show that there are many behavioral instances over time. The actual model for the Wine Club would look something like the diagram in Figure 3.11.

Each of the behavioral entities (wine sales, accessories, and trips) probably would have previously been modeled as part of individual subject areas in separate star schemas or snowflake schemas. In our new model, guess what? They still could be! Nothing we have done so far means that we can't use some dimensional elements if we want to and, more importantly, if we can get the answers we need. Some sharp-eyed readers at this point might be tempted into thinking, "Just hold on a second, what you're proposing for the customer is just some glorified form of common (or conformed) dimension, right?." Well, no.

Figure 3.10 The general model extended to include behavior.

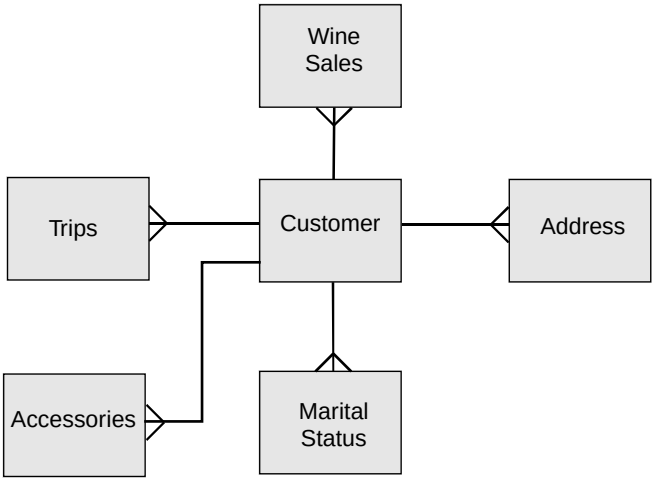

Figure 3.11 The example model extended to include behavior.

There is, of course, some resemblance in this model to the common dimension model that was described earlier on. But remember this: The purpose of a dimension, principally, is to constrain queries against the fact table. The main purpose of a common dimension is to provide a drill across facility from one fact table to another. They are still behavior-centric models. It is not the same thing at all as a model that is designed to be inherently customer centric. The emphasis has shifted away from behavior, and more value is attached to the customer's personal circumstances. This enables us to classify our customers into useful and relevant segments. The difference might seem quite subtle, but it is, nevertheless, significant.

Our general model for a customer centric-data warehouse looks very simple, just three main entity types. Is it complete? Not quite. Remember that there were three main types of customer segmentation. The first two were based on circumstances and behavior. We have discussed these now at some length.

The third type of segment was referred to as a *derived* segment. Examples of derived segments are things like "estimated life time value" and "propensity to churn." Typically, the inclusion and classification of a customer in these segments is determined by some calculation process such as predictive modeling. We would not normally assign a customer a classification in a derived segment merely by assessing the value of some attribute. It is sensible, therefore, to modify our general model to incorporate derived segments, as shown in Figure 3.12.

So this is it. The diagram in Figure 3.12 is the boiled-down general model for a customer centric data warehouse. In theory, it should be able to answer almost any question we might care to throw at it. I say "in theory" because in reality the model will be far more complex than this. We will need to be able to cope with customers whose classification

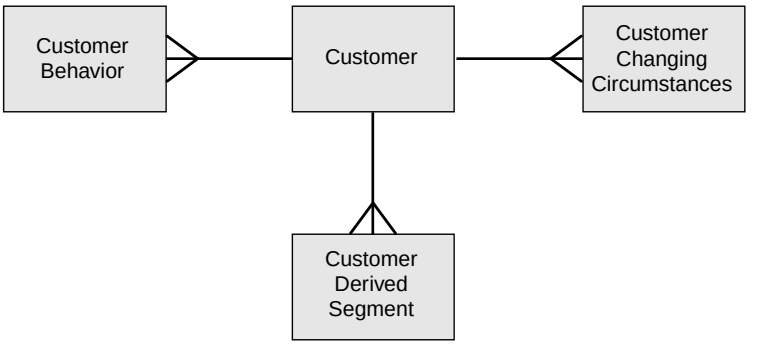

Figure 3.12 General conceptual model for a customer-centric data warehouse.

changes. For example, we might have a derived segment called "life time value" where every customer is allocated an indicator with a value from, say, 1 to 20. Now, Lucie Jones might have a lifetime value indicator of "9." But when Lucie's salary increases, she might be allocated a lifetime value indicator of "10." It might be useful to some companies to invent a new segment called, say, "increasing life time values." This being the case, we may need to track Lucie's lifetime value indicator over time. When we bring time into our segmentation processes, the possibilities become endless. However, the introduction of time also brings with it some very difficult problems, and these will be discussed in the next chapter.

Our model can be described as a general conceptual model (GCM) for a customer-centric data warehouse. The GCM provides us with a template from which all our actual conceptual models in the future can be derived. While we are on the subject of conceptual models, I firmly believe it is high time that we reintroduce the conceptual, logical, and physical data model trilogy into our design process.

Whatever Happened to the Conceptual/Logical/Physical Trilogy?

In the old days there was tradition in which we used a three-stage process for designing a database. The first model that was developed was called the conceptual data model and it was usually represented by an entity relationship diagram (ERD). The purpose of the ERD was to provide an abstraction that represented the data requirements of the organization. Most people with any experience of database design would be familiar with the ERD approach to designing databases.

One major characteristic of the conceptual model is that it ought to be able to be implemented using any type of database. In the 1970s, the relational database was not the most widely used type of DBMS. In those days, the databases tended to be:

1. Hierarchical databases
2. Network databases

The idea was that the DBMS to be ultimately deployed should not have any influence over the way in which the requirements were expressed. So the conceptual data model should not imply the technology to be used in implementing the solution.

Once the DBMS had been chosen, then a logical model would be produced. The logical model was normally expressed as a schema in textual form. So, for instance, where the solution was to be implemented using a relational database, a relational schema would be produced. This consisted of a set of relations, the relationships expressed as foreign key constraints, and a set of domains from which the attributes of the relations would draw their values.

The physical data model consisted of the data definition language (DDL) statements that were needed to actually build, in a relational environment, the tables, indexes, and constraints. This is sometimes referred to as the implementation model.

One of the strengths of the trilogy is that decisions relating to the logical and physical design of the database could be taken and implemented without affecting the abstract model that reflected the business requirements.

The astonishing dominance of relational databases since the mid-1980s has led, in practice, to a blurring of the boundaries between the three models, and it is not uncommon nowadays for a single model to be built, again in the form of an ERD. This ERD is then converted straight into a set of tables in the database. The conceptual model, logical model, and physical model are treated as the same thing. This means that any changes that are made to the design for, say, performance-enhancing reasons are reflected in the conceptual model as well as the physical model. The inescapable conclusion is that the *business requirements are being changed for performance reasons*. Under normal circumstances, in OLTP-type databases for instance, we might be able to debate the pros and cons of this approach because the business users don't ever get near the data model, and it is of no interest to them. They can, therefore, be shielded from it. But data warehouses are different. There can be no debate; the users absolutely have to understand the data in the data warehouse. At least they have to understand that part of it that they use. For this reason, the conceptual data model, or something that can replace its intended role, must be reintroduced as a necessary part of the development lifecycle of data warehouses.

There is another reason why we need to reinvent the conceptual data model for data warehouse development. As we observed earlier, in the past 15 years, the relational database has emerged as a de facto standard in most business applications. However, to use the now well worn phrase, data warehouses are different. Many of the OLAP products are nonrelational, and their logical and physical manifestations are entirely different from the relational model. So the old reasons for having a three-tier approach have returned, and we should respond to this.

The Conceptual Model and the Wine Club

Now that we have the GCM, we can apply its principles to our case study, the Wine Club. We start by defining the information about the customer that we want to store. In the Wine Club we have the following customer attributes:

Customer Information

Title
Name
Address
Telephone number
Date of birth
Sex
Marital status
Childrens' details
Spouse details
Income
Hobbies and interests
Trade or profession
Employers' details

The attributes have to be divided into two types:

1. Attributes that are relatively static (or where previous values can be lost)
2. Attributes that are subject to change

Customer's Static Information

Title
Name
Telephone number
Date of birth
Sex

Customer's Changing Information

Address
Marital status
Childrens' details
Spouse details
Income
Hobbies and interests
Trade or profession
Employers' details

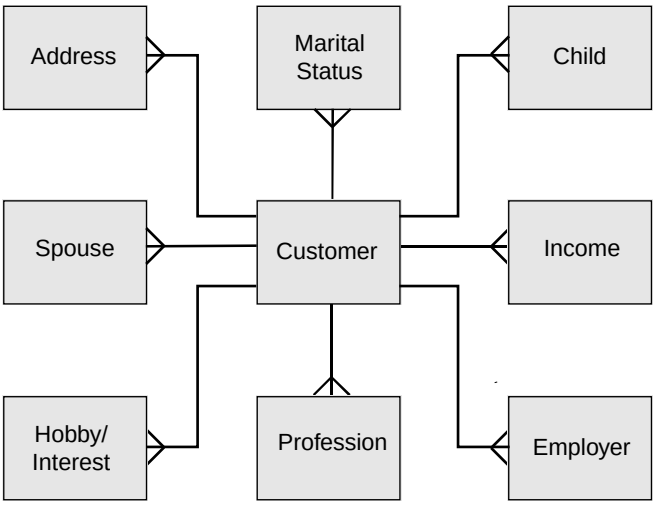

Figure 3.13 Wine Club customer changing circumstances.

We now construct a model like the one in Figure 3.13.

This represents the customer static and changing circumstances. The behavior model is shown in Figure 3.14.

Now, thinking about derived segments, these are likely to be very dynamic in the sense that some derived segments will change over time, some will remain fairly static over time, and others still will appear for a short while and then disappear. Some examples of these, as they apply to the Wine Club, are:

Lifetime value. This is a great way of classifying customers, and every organization should try to do this. It is an example of a fairly static form of segmentation. We would not expect dramatic changes to customers' positions here. It

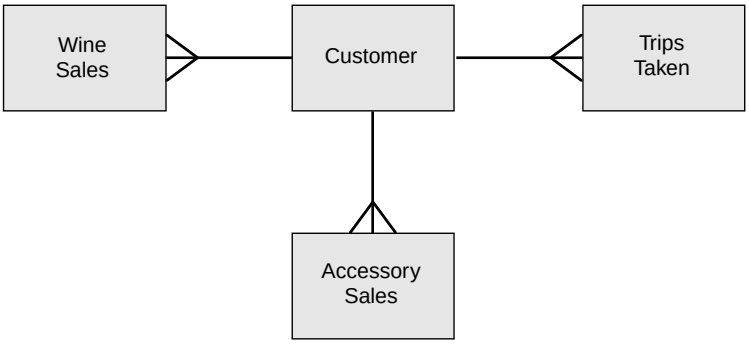

Figure 3.14 Wine Club customer behavior.

would be good to know which customers are on the "generally increasing" and the "generally decreasing" scale.

Recently churned. This is an example of a dynamic classification that will be constantly changing. The ones that we lose that had good lifetime value classifications would appear in our "Win back" derived segment.

Special promotions. These can be good examples where a kind of "one-off" segment can be used effectively. In the Wine Club there would be occasions when, for instance, it needs to sell off a particular product quickly. The requirement would be to determine the customers most likely to buy the product. This would involve examination of previous behavior as well as circumstances (e.g., income category in the case of an expensive wine). The point is that this is a "use once" segment.

Using the three examples above, our derived segments model looks as shown in Figure 3.15.

There is a design issue with segments generally, and that is their general dynamic nature. The marketing organization will constantly want to introduce new segments. Many of them will be of the fairly static and dynamic types that will have long lives in the data warehouse. What we don't want is for the data warehouse administrator to have to get involved in the creation of new tables each time a new classification is invented. This, in effect, results in frequent changes to the data warehouse structure and will lead to the marketing people getting involved in complex change control procedures and might ultimately result in a stifling of creativity. So we need a way of allowing the marketing people to add new derived segments without involving the database administrators too much. Sure, they might need help in expressing the selection criteria, but we don't want to put too many obstacles in their path. This issue will be explored in more detail in Chapter 7—the physical model.

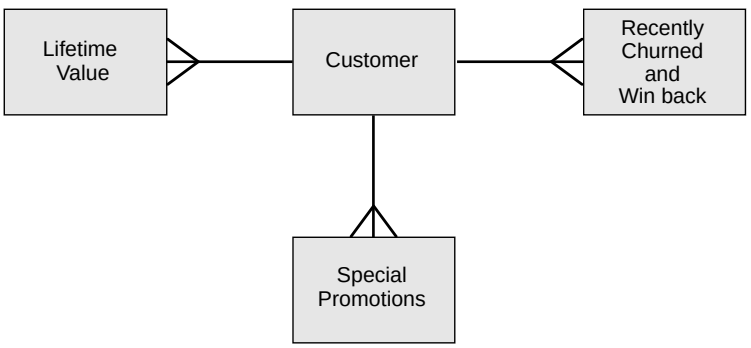

Figure 3.15 Derived segment examples for the Wine Club.

SUMMARY

I have a theory, which is that data warehousing has always been about customer relationships. It's just that previously we didn't entirely make the connection, because, in the early days, CRM had not really been recognized as any kind of discipline at the systems level. The advent of CRM has put the spotlight firmly back on data warehousing.

Data warehouses provide the technology to enable us to perform customer relationship analysis. The management of the relationship is the value that is added by the business. This is the decision-making part. The warehouse is doing what it has always done, providing decision support. That is why this book is about supporting customer relationship management.

In this chapter we have been looking at some of the design issues and tried to quantify and rationalize some aspects of data warehouse design that have been controversial in the past. There are good reasons for using dimensional schemas, but there are cases where they can work against us. The best solution for CRM is to use dimensional models where they absolutely do add value, in modeling customers' behavior, but to use more "conventional" approaches when modeling customers' circumstances.

Toward the end of the chapter we developed a general conceptual model (Figure 3.12) for a customer-centric data warehouse. We will develop this model later on in this book by applying some of the needs of our case study, the Wine Club.

Firstly, however, the awfully thorny subject of time has to be examined.