Chapter 1: What is a Data Model?

Overview

How do I get there?
Maps, blueprints, data models
Please show me the way

I gave the steering wheel a heavy tap with my hands as I realized that once again, I was completely lost. It was about an hour before dawn, I was driving in France, and an important business meeting awaited me. I spotted a gas station up ahead that appeared to be open. I parked, went inside, and showed the attendant the address of my destination.

I don't speak French and the attendant didn't speak English. The attendant did, however, recognize the name of the company I needed to visit. Wanting to help and unable to communicate verbally, the attendant took out a pen and paper. He drew lines for streets, circles for roundabouts along with numbers for exit paths, and rectangles for his gas station and my destination, MFoods. The picture he drew resembled that which appears in Figure 1.1.

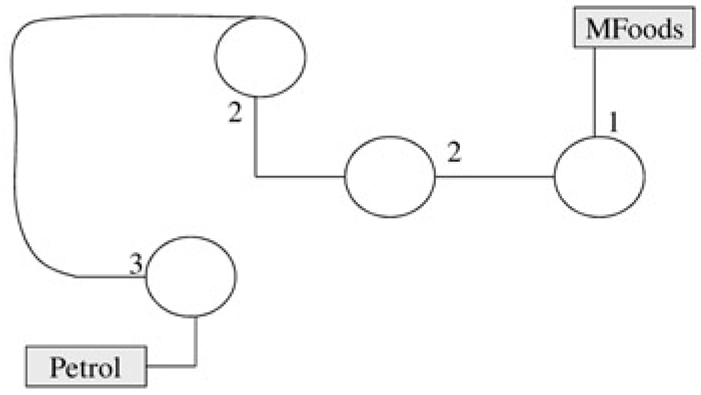


Figure 1.1: Simplification of geographic landscape

With this custom-made map, which contained only the information that was relevant to me, I arrived at my address without making a single wrong turn. This map was a model of the actual roads I needed to travel.

A map simplifies a complex *geographic* landscape in the same way that a data model simplifies a complex *information* landscape. This chapter explains the data model and its role as an invaluable wayfinding tool. It also introduces the ice cream and business card examples which are carried throughout the text.

Wayfinding Explained

If the term 'data model' does not excite you or your business users, try the term 'wayfinding' instead. Wayfinding encompasses all of the techniques and tools used by people and animals to find their way from one site to another. If travelers navigate by the stars, for example, the stars are their wayfinding tools. Maps and compasses are also wayfinding tools.

All models are wayfinding tools. A model is a set of symbols and text used to make a complex concept easier to grasp. The world around us is full of obstacles that can overwhelm our senses and make it very challenging to focus only on the relevant information needed to make intelligent decisions. A map helps a visitor navigate a city. An organization chart helps an employee understand reporting relationships. A blueprint helps an architect communicate building plans. The map, organization chart, and blueprint are all types of models that represent a filtered, simplified view of something complex, with the goal of improving a wayfinding experience by helping people understand part of the real world.

It would probably have taken me hours of trial and error to reach my destination in France, whereas that simple map the gas station attendant drew provided me with an almost instantaneous broad understanding of how to reach my destination. A model makes use of standard symbols that allow one to grasp the content quickly. In the map he drew for me, the attendant used lines to symbolize streets and circles to symbolize roundabouts. His skillful use of those symbols helped me visualize the streets and roundabouts.

Data Model Explained

When I was in college, the term 'information overload' was used to mean our brains had reached the maximum amount of words spoken by the professor, appearing on her flipcharts, and in the page of notes in front of me. It was time for a stroll around campus, a game of tennis, or a couple of quarters in Space Invaders to get my mind recharged and ready for more. Today however, it seems we are creating and receiving more and more information, and taking fewer and fewer breaks. I have heard it quoted several times that the amount of information in the world is increasing by over 60% per year! I shudder to myself, wondering what very small portion of all of this information we really, truly understand.

Luckily, there is a tool that can help simplify all of this information - the data model. A data model is a wayfinding tool for both business and IT professionals, which uses a set of symbols and text to precisely explain a subset of real information to improve communication within the organization and thereby lead to a more flexible and stable application environment. A line represents a motorway on a map of France. A box with the word 'Customer' within it represents the concept of a real Customer such as Bob, IBM, or Walmart on a data model.

In other words, a map simplifies a complex *geographic* landscape in the same way that a data model simplifies a complex *information* landscape. In many cases, the complexities in the actual data can make those roundabouts in France look ridiculously simple.

Our broad definition of a data model as a set of symbols and text which precisely explain a subset of real information encompasses models in many different forms. Data models can look like the box and line drawings which are the subject of this book, or they can take other forms, such as Unified Modeling Language (UML) Class Diagrams, spreadsheets, or State Transition

Diagrams. All of these models are wayfinding tools designed with the single purpose of simplifying complex information in our real world.

Fun with Ice Cream

Perhaps the most common form of data model we work with on a daily basis is the spreadsheet. A spreadsheet is a representation of a paper worksheet, containing a grid defined by rows and columns, where each cell in the grid can contain text or numbers. The columns often contain different types of information. For example, I recently returned from a trip to Rome and I loved their ice cream (gelato). Upon entering a gelato store, you will see several spreadsheets. One example is shown in Table 1.1 which is a listing of ice cream. Table 1.2 contains the ice cream sizes along with prices (in Euros).

Table 1.1: Sample ice cream flavors

Banana
Cappuccino
Chocolate
Chocolate Chip
Coffee
Kiwi
Marshmallow
Pistachio
Strawberry
Vanilla

Table 1.2: Ice cream sizes with prices

1 Scoop	1.75
2 Scoops	2.25
3 Scoops	2.60

This is a data model because it is a set of symbols (in this case, text) that are used to describe something real in our world (in this case, the yummy ice cream flavors along with prices). Guess how many scoops of chocolate gelato I purchased?

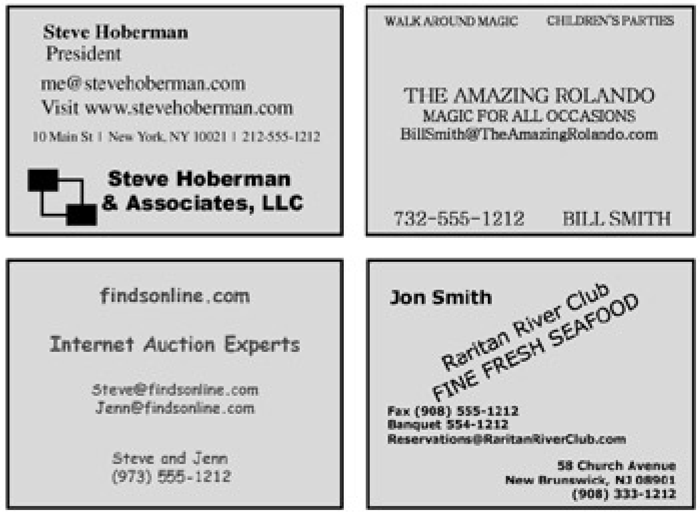
The data model format that is the subject of this book is very similar to a spreadsheet. (Although the definition of 'data model' is broader, going forward, when I use the term 'data model', I am referring to the format which is the subject of this book.) Unlike a spreadsheet however, a data model:

- **Contains only types.** Data models don't usually display actual values such as 'Chocolate' and '3 Scoops'. Data models display concepts or types. So, a data model would display the type **Ice Cream Flavor**, instead of showing the actual values 'Chocolate' or 'Vanilla'. A data model would display the type **Ice Cream Size**, instead of showing the actual values '1 Scoop' or '2 Scoops'.
- **Contains interactions.** Data models capture how concepts interact with each other. For example, what is the interaction between **Ice Cream Flavor** and **Ice Cream Size**? If one orders three scoops, for example, must they all be the same flavor or could you have three different flavors? Interactions such as those between **Ice Cream Flavor** and **Ice Cream Size** are represented on a data model.
- **Provides a concise communication medium.** A single sheet of paper containing a data model communicates much more than a single piece of paper containing a spreadsheet. Data models display types, not actual values, and they use simple yet powerful symbols to communicate interactions. We can capture all of the types and interactions within the ice cream example in a much more concise format using a data model rather than in a spreadsheet.

Fun with Business Cards

Business cards contain a wealth of data about people and the companies for which they work. In this book, I illustrate many data modeling concepts by using business cards as the basis for a model. By building a business card data model, we see firsthand how much knowledge we gain from the actual values on a business card or, in a broader sense, the contact management area.

I opened the drawer in my nightstand (a scary proposition, as it had not been cleaned since the mid-1980s) and grabbed a handful of business cards. I laid them out and picked four that I thought would be the most fun to model. I chose my current business card, a business card from an internet business that my wife and I tried to start years ago when dot-com was hot, a business card from a magician who performed at one of our parties, and a business card from one of our favorite restaurants. I changed the names and contact information to protect the innocent, and reproduced them here, in Figure 1.2.


Steve Hoberman
President
me@stevehoberman.com
Visit www.stevehoberman.com
10 Main St | New York, NY 10021 | 212-555-1212
Steve Hoberman
& Associates, LLC

WALK AROUND MAGIC CHILDREN'S PARTIES
THE AMAZING ROLANDO
MAGIC FOR ALL OCCASIONS
BillSmith@TheAmazingRolando.com
732-555-1212 BILL SMITH

findsonline.com
Internet Auction Experts
Steve@findsonline.com
Jenn@findsonline.com
Steve and Jenn
(973) 555-1212

Jon Smith
Raritan River Club
FINE FRESH SEAFOOD
Fax (908) 555-1212
Banquet 554-1212
Reservations@RaritanRiverClub.com
58 Church Avenue
New Brunswick, NJ 08901
(908) 333-1212

Figure 1.2: Four business cards from my nightstand

What information do you see on these business cards?

Assuming our objective with this exercise is to understand the information on the business cards, with an end goal of building a successful contact management application, let's begin by listing some of this information:

- Steve Hoberman & Associates, LLC
  BILL SMITH
  Jon Smith
  212-555-1212
  MAGIC FOR ALL OCCASIONS
  Steve and Jenn
  58 Church Avenue
  FINE FRESH SEAFOOD
  President

We quickly realize that even though we are dealing with only four business cards, listing all the data would do little to aid our understanding. Now, imagine that instead of limiting ourselves to just these four cards, we looked through all the cards in my nightstand—or worse yet, every business card that has ever been received! We would become overloaded with data quickly.

A data model groups data together to make them easier to understand. For example, we would examine the following set of data and realize that they fit in a group (or spreadsheet column heading) called **Company Name**:

- Steve Hoberman & Associates, LLC
  The Amazing Rolando
  findsonline.com
  Raritan River Club

Another spreadsheet column heading could be **Phone Number**. Table 1.3 captures this subset of the business card information in the form of a spreadsheet.

Table 1.3: Subset of business card information in a spreadsheet format
➡Open table as spreadsheet

	Company	**Phone number**
Business card 1	Steve Hoberman & Associates, LLC	212-555-1212
Business card 2	findsonline.com	973-555-1212
Business card 3	The Amazing Rolando	732-555-1212
Business card 4	Raritan River Club	(908)333-1212 (908)555-1212 554-1212

Taking this exercise a step further, we can organize the data on the cards into the following groups:

- Person name
- Person title
- Company name
- Email address
- Web address
- Mailing address
- Phone number
- Logo (the image on the card)
- Specialties (such as "MAGIC FOR ALL OCCASIONS")

So, are we finished? Is this listing of groups a data model? Not yet. We are still missing a key ingredient: the interactions or relationships between these groups. For example, what is the interaction between **Company Name** and **Phone Number**? Can a Company have more than one **Phone Number**? Can a **Phone Number** belong to more than one Company? Can a Company exist without a **Phone Number**? These questions, and others, need to be asked and answered during the process of building the data model.

In order to build any wayfinding tool, one must get lost enough times to know the right path. For example, the first person who builds a map of a region must have taken quite a bit of time and made quite a few wrong turns before completing the map. The process of building a map is both challenging and time-consuming.

The same is true for the process of completing a data model. There is the 'data model' and then there is 'data modeling'. Data modeling is the process of building a data model. More specifically, data modeling is the set of techniques and activities that enable us to capture the structure and operations of an organization, as well as the proposed information solution that will enable the organization to achieve its goals. The process requires many skills, such as listening ability, courage to ask lots of questions, and even patience. The data modeler needs to speak with individuals from many different departments with varying levels of technical and business experiences and skills. The data modeler not only needs to understand these individuals' views of their world, but also be able to demonstrate this understanding through feedback during the conversation and also as a final artifact in the form of the model. At the beginning of a project, it is rare that you, as the data modeler, are handed all of the information you need to complete the model. It will require reading through lots of documentation and asking hundreds of business questions.

Exercise 1: Educating Your Neighbor

1. Reinforce your own understanding of what a data model is by explaining the concept of a data model to someone completely outside the world of IT, such as to a neighbor, family member or friend. ?

Did they get it?

Answers

1. I find the analogy that I use most frequently is comparing the data model to a blueprint. Most non-technical friends, family, and neighbors understand this analogy. "Just like you need a blueprint to ensure a sound building structure, you need a data model to ensure a sound application." Sometimes I also explain to people that a data model is nothing more than a fancy spreadsheet, which contains not just the spreadsheet columns, but also the business rules binding these columns. If both the blueprint and spreadsheet analogies fail, I quickly change the subject to the other person and ask what they do (and hope they never ask me again!).

Refer to the Appendix to see how I explain the concept of a data model.

Key Points

- Wayfinding encompasses all of the techniques and tools used by people and animals to find their way from one site to another.
- A data model is a wayfinding tool, for both business and IT professionals, which uses a set of symbols and text to precisely explain a subset of real information to improve communication within the organization, and thereby lead to a more flexible and stable application environment.
- Data models come in many different forms. The most common and globally-understood form is a spreadsheet.
- The data model format that is the subject of this book is similar to the spreadsheet, yet is type-based, contains interactions, and is extensible.
- Data modeling is the process of building the data model. This process requires many non-technical skills, such as listening ability, courage to ask lots of questions, and patience.

Chapter 2: Why Do We Need a Data Model?

Overview

Ambiguous talk
Data models are precise
Zero, one, many

Data modeling is an essential part of building an application. Communication and precision are the two key benefits that make a data model so important. This chapter explains these two core benefits, followed by a description of the areas within the business and application where the data model can be used. You will learn where communication occurs and about the three situations that can weaken data model precision.

Communication

People with different backgrounds and levels of experience across departments and functional areas need to speak with each other about business concerns and to make business decisions. In a conversation, therefore, there is a need to know how the other party views concepts such as Customer or Sales. The data model is an ideal tool for understanding, documenting, and eventually reconciling different perspectives.

When the spoken word failed to reach me, the model that the gas station attendant drew for me clearly explained how to get to my destination. Regardless of whether we are trying to understand how key concepts in a business relate to one another or the workings of a 20-year-old order-processing system, the model becomes an ideal mechanism for explaining information.

Data models allow us to communicate the same information at different levels of detail. For example, I recently built a data model to capture consumer interactions within snack food. So if someone called up a company and complained about one of the company's products, the model I built would store this complaint and related information about it. The key business user and I built a very high-level data model showing the subjects that are relevant for the project. The model helped with scoping the project, understanding key terms, and building a rapport with the business. This model became the mechanism we used to bridge our understandings of key terms such as Consumer, Product, and Interaction. Several months later, I used a much more detailed model of the same consumer-interaction information to inform the report developers of exactly what was expected to appear on each report along with all the necessary selection criteria.

The communication we derive from modeling does not begin when the data modeling phase has ended. That is, much communication and knowledge is shared during the process of building the model. The means are just as valuable as the end. Let's look at the communication benefits derived both during and after the modeling process in more detail.

Communicating During the Modeling Process

During the process of building data models, we are forced to analyze data and data relationships. We have no choice but to acquire a strong understanding of the content of what is being modeled. A lot of knowledge is gained as the people involved in the modeling process challenge each other about terminology, assumptions, rules, and concepts.

During the process of modeling a recipe management system for a large manufacturing company, I was amazed to witness team members with years of experience debate whether the concept of an Ingredient differed from the concept of Raw Material. After a 30 minute discussion on ingredients and raw materials, everyone who participated in this modeling effort benefited from the debate and left the modeling session with a stronger understanding of recipe management.

When we model the business card example, you'll see that we can learn a lot about the person, company, and contact management in general, during the modeling process.

Communicating After the Modeling Process

The completed data model is the basis for discussing what to build in an application, or more fundamentally, how something works. The data model becomes a reusable map to which analysts, modelers, and developers can refer to understand how things work. In much the same way as the first mapmaker painfully learned and documented a geographic landscape for others to use for navigation, the modeler goes through a similar exercise (often painful, but in a good way) so that others can understand an information landscape.

Before I started working at a large manufacturing company, my soon-to-be manager gave me a large book containing a set of data models for the company. I read this book several times, becoming familiar with the key concepts in the business and their business rules. On my first day on the job, I already knew much about how the business worked. When my colleagues mentioned terms specific to the company, I already knew what they meant.

In our example with the business cards, once we complete the model, others can read it to learn about contact management.

Precision

Precision with respect to data modeling means that there is a clear, unambiguous way of reading every symbol and term on the model. You might argue with others about whether the rule is accurate, but that is a different argument. In other words, it is not possible for you to view a symbol on a model and say, "I see A here" and for someone else to view the same symbol and respond, "I see B here."

Going back to the business card example, let's assume we define a 'contact' to be the person or company that is listed on a business card. Someone states that *a contact can have many phone numbers*. This statement is imprecise, as we do not know whether a contact can exist without a phone number, must have at least one phone number, or must have many phone numbers. Similarly, we do not know whether a phone number can exist without a contact, must belong to only one contact, or can belong to many contacts. The data model introduces precision, such as converting this vague statement into these assertions:

- Each contact must be reached by one or many phone numbers.
- Each phone number must belong to one contact.

Because the data model introduces precision, valuable time is not wasted trying to interpret the model. Instead, time can be spent debating and then validating the concepts on the data model.

There are three situations however, that can degrade the precision of a data model:

- **Weak definitions.** If the definitions behind the terms on a data model are poor or nonexistent, multiple interpretations of terms becomes a strong possibility. Imagine a business rule on our model that states that an employee must have at least one benefits package. If the definition of Employee is something meaningless like "An Employee is a carbon-based life form", we may wrongly conclude that this business rule considers both job applicants and retired employees to be employees.
- **Dummy data.** The second situation occurs when we introduce data that are outside the normal set of data values we would expect in a particular data grouping. An old fashioned trick for getting around the rigor of a data model is to expand the set of values that a data grouping can contain. For example, if a contact must have at least one phone number and for some reason, a contact arrives in the application with no phone numbers, one can create a fake phone number such as 'Not Applicable' or '99' or 'other' and then the contact can be entered. In this case, adding the dummy data allows a contact to exist without a phone number, which violates, but circumvents our original business rule.
- **Vague or missing labels.** A model is read in much the same way as a book is read, with proper sentence structure. A very important part of the sentence is the verbs. On a data model, these verbs are captured when describing how concepts on the model relate to each other. Concepts like Customer and Order for example, may relate to each other through the verb 'place'. That is "A Customer can place one or many Orders." Vague verbs such as 'associate' or 'have', or missing verbs altogether, reduce the precision of the model, as we cannot accurately read the sentences.

In a data model, precision is also the result of applying a standard set of symbols. The traffic circles the gas station attendant drew for me were standard symbols that we both understood. There are also standard symbols used in data models, as we will discover shortly.

Data Model Uses

Traditionally, data models have been built during the analysis and design phases of a project to ensure that the requirements for a new application are fully understood and correctly captured before the actual database is created. There are, however, other uses for modeling than simply building databases. Among the uses are the following:

- **To understand an existing application.** The data model provides a simple and precise picture of the concepts within an application. We can derive a data model from an existing application by examining the application's database and building a data model of its structures. The technical term for the process of building data models from existing applications is 'reverse engineering'. The trend in many industries is to buy more packaged software and to build less internally; therefore our roles as modelers are expanding and changing. Instead of modeling a new application, the data modeler may capture the information in existing systems, such as packaged software. In fact, modeling of existing systems frequently takes place because originally, when the application was built, no modeling was done and therefore, the system is running with an inferior design that few people understand. Recently, a manufacturing organization needed to move a 25 year old application to a new database platform. This was a very large application, so to understand its structures, we built a data model of it.
- **To perform impact analysis.** A data model can capture the concepts and interactions that are impacted by a development project or program. What is the impact of adding or modifying structures for an application already in production? How many of an application's structures are needed for archival purposes? Many organizations today purchase software and then customize it. One example of impact analysis would be to use data modeling to determine what impact modifying its structures would have on the purchased software.
- **To understand a business area.** As a prerequisite to a large development effort, it usually is necessary to understand how the business works before you can understand how the applications that support the business will work. Before building an order entry system, for example, you need to understand the order entry business process. Data modeling provides a formal process for learning how part of the business works. The completed model will describe the business area. That is, the completed data model becomes a wayfinding tool for the order data entry area.
- **To educate team members.** When new team members need to come up to speed or developers need to understand requirements, a data model is an effective explanatory medium. A picture is worth a thousand words, and a data model is a picture that can convey information at different levels of detail. Whenever a new person joined our area, I spent some time walking through a series of high-level data models to educate the person on concepts and rules as quickly as possible.

Exercise 2: Converting the Non-Believer

Find someone in your organization who is a data model non-believer and try to convert him or her.

What obstacles did you run into? Did you overcome them?

Key Points

- The two main benefits of a data model are communication and precision.
- Communication occurs both during the building of the data model and after its completion.
- Data model precision can be compromised by weak definitions, dummy data, and vague or missing labels.
- Communication and precision make a data model an excellent tool for building new applications.
- Other uses for data models include understanding existing applications and business areas, performing impact analysis, and educating team members.

Chapter 3: What Camera Settings Also Apply to a Data Model?

Overview

Cameras have settings
Zoom, Focus, Timer, Filter
Data models, too

This chapter compares the data model to a camera, exploring four settings on the camera that equate perfectly to the data model. Understanding the impact these settings can have on a data model will increase the chances for a successful application. This chapter also compares the camera's film to the three levels at which the data model can exist: subject area, logical, and physical.

The Data Model and the Camera

A camera has many settings available to take the perfect picture. Imagine facing an awesome sunset with your camera. With the same exact sunset, you can capture a very different image based on the camera's settings, such as the focus, timer, and zoom. You might for example zoom out to capture as much of the sunset as possible, or zoom in and focus on people walking by with the sunset as a backdrop. It depends on what you want to capture in the photograph.

There are four settings on a camera that translate directly over to the data model: zoom, focus, timer, and filter. A model is characterized by one value from each setting. See Figure 3.1.

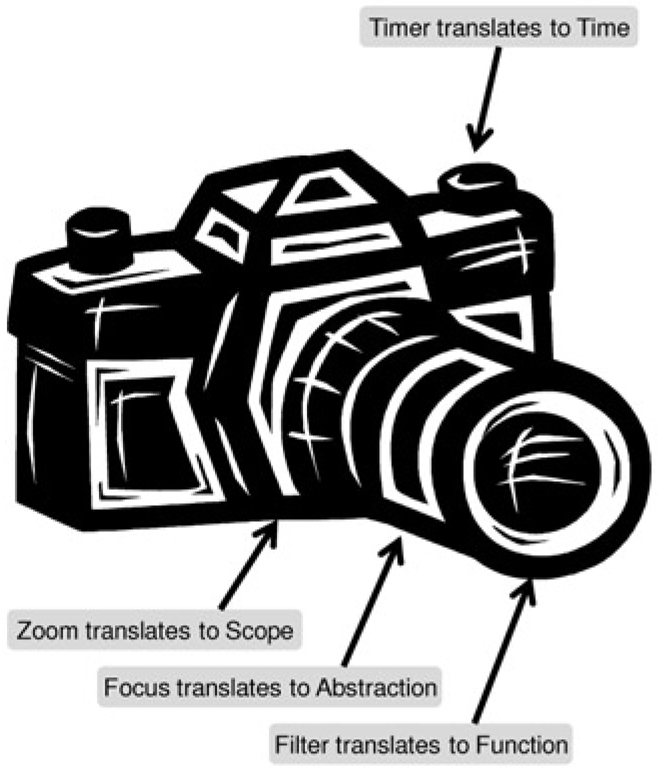

Timer translates to Time
Zoom translates to Scope
Focus translates to Abstraction
Filter translates to Function

Figure 3.1: Camera settings that translate to data model variations

The zoom setting on the camera allows the photographer to capture a broad area with minimal detail, or a narrow scope but with more detail. Similarly, the scope setting for the model varies how much you see in the picture. The focus setting on the camera can make certain objects appear sharp or blurry. Similarly, the abstraction setting for the model can use generic concepts such as Party and Event to "blur" the distinction between concepts. The timer allows for a real-time snapshot or a snapshot for some time in the future. Similarly, the time setting for the model can capture a current view or a "to be" view sometime in the future. The filter setting can adjust the appearance of the entire picture to produce a certain effect. Similarly, the function setting for the model adjusts the model with either a business or application view.

And don't forget that the type of film you use is important! A proof sheet shows all of the images on a single piece of paper, the negative has the raw format of the image, and the output can be in any one of a number of formats, including paper film, slide, or digital. Similarly, the same information image can exist at a subject area, logical, or physical level of detail on a data model.

Which setting is right for your model? As with photographing the sunset, it depends on what you want to capture. Match the goals of your model with the appropriate model settings to improve the overall quality of the data model and the application it supports.

Scope

Both a data model and photograph have boundaries. Boundaries determine what will be shown. A photograph can capture my youngest daughter enjoying ice cream (actually her whole face enjoying the ice cream), or the photograph can capture my daughter and her surroundings, such as the ice cream shop. Similarly, the data model can include just claims processing, or it can include all concepts in the insurance business. Typically, the scope of a data model is a department, organization or industry:

- **Department (Project).** The most common type of modeling assignment has project-level scope. A project is a plan to complete a software development effort, often defined by a set of deliverables with due dates. Examples include a sales data mart, broker trading application, reservations system, and an enhancement to an existing application.
- **Organization (Program).** A program is a large, centrally organized initiative that contains multiple projects. It has a start date and, if successful, no end date. Programs can be very complex and require long-term modeling assignments. Examples include a data warehouse, operational data store, and a customer relationship management system.
- **Industry.** An industry initiative is designed to capture everything in an industry, such as manufacturing or banking. There is much work underway in many industries to share a common data model. Industries such as health care and telecommunications have consortiums where common data modeling structures are being developed. Having such a common structure makes it quicker to build applications and easier to share information across organizations within the same industry.

Abstraction

A photograph can be blurry or in focus. Similar to how the focus on a camera allows you to make the picture sharp or fuzzy, the abstraction setting for a model allows you to represent "sharp" (concrete) or "fuzzy" (generic) concepts.

Abstraction brings flexibility to your data models by redefining and combining some of the data elements, entities, and relationships within the model into more generic terms. Abstraction is the removal of details in such a way as to broaden applicability to a wider class of situations, while preserving the important properties and essential nature of concepts or subjects. By removing these details, we remove differences and, therefore, change the way we view these concepts or subjects, including seeing similarities that were not apparent or even existent before. For example, we may abstract Employee and Consumer into the more generic concept of Person. A Person can play many Roles, two of which are Employee and Consumer. The more abstract a data model, the fuzzier it becomes.

On a data model, concepts can be represented at different levels of abstraction: 'in the business clouds', 'in the database clouds', or 'on the ground':

- **In the business clouds.** At this level of abstraction, only generic *business* terms are used on the model. The business clouds model hides much of the real complexity within generic concepts such as Person, Transaction, and Document. In fact, both a candy company and insurance company can look very similar to each other using business cloud concepts. If you lack business understanding or do not have access to business documentation and resources, a model 'in the business clouds' can work well.

- **In the database clouds.** At this level of abstraction, only generic *database (db)* terms are used across the model. The database clouds model is the easiest level to create, as the modeler is "hiding" all of the business complexity within database concepts such as Entity, Object, and Attribute. If you have no idea how the business works and you want to cover all situations for all types of industries, a model 'in the database clouds' can work well.
- **On the ground.** This model uses a minimal amount of business and database cloud entities, with a majority of the concepts representing concrete business terms such as Student, Course, and Instructor. This model takes the most time to create of the three varieties. It also can add the most value towards understanding the business and resolving data issues.

Time

Most cameras have a timer, allowing the photographer to run quickly and get in the picture. Similar to how the timer on a camera allows you to photograph a current or future scene, the time setting for a model allows you to represent a current or future "to be" view on a model.

A model can represent how a business works today or how a business might work sometime in the future:

- **Today.** A model with the today setting captures how the business works today. If there are archaic business rules, they will appear on this model, even if the business is planning on modifying them in the near future. In addition, if an organization is in the process of buying another company, selling a company, or changing lines of business, a today view would not show any of this. It would only capture an 'as is' view.
- **Tomorrow.** A model with the tomorrow setting can represent any time period in the future, and is usually an idealistic view. Whether end of the year, five years out, or 10 years out, a tomorrow setting represents where the organization wants to be. When a model needs to support an organization's vision or strategic view, a tomorrow setting is preferred. I worked on a university model that represented an end of year view, as that would be when a large application migration would be completed. Note that most organizations who need a tomorrow view have to first build a today view to create a starting poi

Function

Filters are plastic or glass covers that, when placed over the camera lens, adjust the picture with the color of the filter, such as making the picture more bluish or greenish. Similar to how a filter on a camera can change the appearance of a scene, the function setting for a model allows you to represent either a business or functional view on the model.

Are we modeling the business' view of the world or the application's view of the world? Sometimes they can be the same and sometimes they may be very different:

- **Business.** This filter uses business terminology and rules. The model represents an application-independent view. It does not matter if the organization is using a filing cabinet to store its information, or the fastest software system out there; the information will be represented in business concepts.
- **Application.** This filter uses application terminology and rules. It is a view of the business through the eyes of an application. If the application uses the term 'Object' for the term 'Product', it will appear as 'Object' on the model and it will be defined according to the way the application defines the term, not how the business defines it.

Format

A camera has a number of different formats in which the photo can be captured. The format setting adjusts the level of detail for a model, making the model either at a very broad and high level subject area view, or a more detailed logical or physical view:

- **Subject area.** Often when a roll of film is processed, a proof sheet containing small thumbnail images of each photograph is included. The viewer can get a bird's eye view of all of the photographs on a single sheet of photo paper. This bird's eye view is analogous to the subject area model (SAM). A SAM represents the business at a very high level. It is a very broad view containing only the basic and critical concepts for a given scope. Here, basic means that the subject area is usually mentioned a hundred times a day in normal conversation. Critical means that without this subject area, the department, company, or industry would be greatly changed. Some subject areas are common to all organizations, such as Customer, Product, and Employee. Other subject areas are very industry or department specific, such as Policy for the insurance industry or Trade for the brokerage industry.
- **Logical.** Before the days of digital cameras, a roll of processed film would be returned with a set of negatives. These negatives represented a perfect view of the picture taken. The negative corresponds to the logical data model. A logical data model (LDM) represents a detailed business solution. It is how the modeler captures the business requirements without complicating the model with implementation concerns such as software and hardware.
- **Physical.** Although a negative is a perfect view of what was taken through the camera, it is not very practical to use. You can't, for example, put a negative in a picture frame or in a photo album and easily share it with friends. You need to convert or 'instantiate' the negative into a photograph or slide or digital image. Similarly, the logical data model usually needs to be modified to make it usable. Enter the physical data model (PDM), which is the 'incarnation' or 'instantiation' of the LDM, the same way as the photograph is the 'incarnation' of the negative. A PDM represents a detailed technology solution. It is optimized for a specific context (such as specific software or hardware). A physical data model is the logical data model modified with performance-enhancing techniques for the specific environment in which the data will be created, maintained, and accessed.

Exercise 3: Choosing the Right Setting

1. In the following table, check off the most appropriate settings for each of these scenarios. See the Appendix for my answers.

1. Explain to a team of developers how an existing contact management application works

   ➡Open table as spreadsheet

Scope	**Abstraction**	**Time**	**Function**
❑ Dept	❑ Bus clouds	❑ Today	❑ Bus
❑ Org	❑ DB clouds	❑ Tomorrow	❑ App
❑ Industry	❑ On the ground		

2. Explain the key manufacturing concepts to a new hire

   ➡Open table as spreadsheet

Scope	**Abstraction**	**Time**	**Function**
❑ Dept	❑ Bus clouds	❑ Today	❑ Bus
❑ Org	❑ DB clouds	❑ Tomorrow	❑ App
❑ Industry	❑ On the ground		

3. Capture the detailed requirements for a new sales datamart (A datamart is a repository of data that is designed to meet the needs of a specific set of users)

   ➡Open table as spreadsheet

Scope	**Abstraction**	**Time**	**Function**
❑ Dept	❑ Bus clouds	❑ Today	❑ Bus
❑ Org	❑ DB clouds	❑ Tomorrow	❑ App
❑ Industry	❑ On the ground		

Answers

1. In the following table, I checked off the most appropriate settings for each of these scenarios.

1. Explain how a contact management legacy application works to a team of developers

   ➡Open table as spreadsheet

Scope	**Abstraction**	**Time**	**Function**
☒Dept	☒Bus clouds	☒Today	❑ Bus
❑ Org	❑ DB clouds	❑ Tomorrow	☒App
❑ Industry	❑ On the ground		

2. Explain the key manufacturing concepts to a new hire

➡Open table as spreadsheet

Scope	**Abstraction**	**Time**	**Function**
❑ Dept	❑ Bus clouds	☒Today	☒Bus
☒Org	❑ DB clouds	❑ Tomorrow	❑ App
❑ Industry	☒On the ground		

3. Capture the detailed requirements for a new sales datamart

➡Open table as spreadsheet

Scope	**Abstraction**	**Time**	**Function**
☒ Dept	☒ Bus clouds	☒ Today	❑ Bus
❑ Org	❑ DB clouds	❑ Tomorrow	☒ App
❑ Industry	❑ On the ground		

Key Points

- There are four settings on a camera that translate directly to the model: zoom, focus, timer, and filter. Zoom translates into data model scope, focus into the level of abstraction, timer into whether the data model is capturing an 'as is' or future view, and filter into whether the model is capturing a business or application perspective.
- Match the goals of your model with the appropriate model settings to improve the overall quality of the data model and resulting application.
- Don't forget the film options! Would your audience prefer to view the proof sheet (subject area model), the negative (logical data model), or the photograph (physical data model)?

Section II: Data Model Components

Chapter List

Part Overview

And then each Customer can buy many Products, and they lived happily ever after.
CUSTOMER
BUY
PRODUCT

Section II explains all of the symbols and text on a data model. Chapter 4 explains entities, Chapter 5 is about data elements, Chapter 6 discusses relationships, and Chapter 7 is on keys. By the time you finish this section, you will be able to 'read' a data model of any size or complexity.

Chapter 4 defines an entity and discusses the different categories of entities. Entity instances are also defined. The three different levels at which entities may exist, subject area, logical, and physical, are also explained, as well as the concepts of a weak verse strong entity.

Chapter 5 defines a data element and discusses domains. Examples are provided for the three different types of domains.

Chapter 6 defines rules and relationships. Data rules are distinguished from action rules. Cardinality and labels are explained so that the reader can read any data model as easily as reading a book. Other types of relationships, such as recursive relationships, and subtyping are discussed, as well.

Chapter 7 defines keys and distinguishes the terms candidate, primary, alternate key. Surrogate keys and foreign keys are also defined, along with a discussion on their importance.

Chapter 4: What are Entities?

Overview

Concepts of interest
Who, What, When, Where, Why, and How
Entities abound

As I walked around the room to see if any students had questions, I noticed someone in the last row had already finished the exercise. I walked over to where she was sitting and looking over her shoulder, noticed only a handful of boxes on the page. The large box in the center contained the word 'Manufacturing'. I asked her for her definition of 'Manufacturing'. "Manufacturing is the production process of how we turn raw materials into finished goods. All the manufacturing steps are in this box."

The data model boxes (also known as 'entities'), however, are not designed to represent or contain processes. Instead, they represent the concepts that are used *by* the processes. The Manufacturing entity on her model was eventually transformed into several other entities, including Raw Material, Finished Goods, Machinery, and Production Schedule.

This chapter defines the concept of an entity and discusses the different categories (Who, What, When, Where, Why and How) of entities. Entity instances are also defined. The three different levels of entities, subject area, logical, and physical, are also explained, as well as the concepts of a weak versus a strong entity.

Entity Explained

An entity represents a collection of information about something that the business deems important and worthy of capture. A noun or noun phrase identifies a specific entity. It fits into one of several categories - who, what, when, where, why, or how. Table 4.1 contains a definition of each of these entity categories along with examples.

Table 4.1: Definitions and examples of entity categories

Open table as spreadsheet

Category	Definition	Examples
Who	Person or organization of interest to the enterprise. That is, "*Who* is important to the business?" Often a 'who' is associated with a role such as Customer or Vendor.	Employee, Patient, Gambler, Suspect, Customer, Vendor, Student, Passenger, Competitor
What	Product or service of interest to the enterprise. It often refers to what the organization makes that keeps it in business. That is, "*What* is important to the business?"	Product, Service, Raw Material, Finished Good, Course, Song, Photograph
When	Calendar or time interval of interest to the enterprise. That is, "*When* is the business in operation?"	Time, Date, Month, Quarter, Year, Calendar, Semester, Fiscal Period, Minute
Where	Location of interest to the enterprise. Location can refer to actual places as well as electronic places. That is, "*Where* is business conducted?"	Mailing Address, Distribution Point, Website URL, IP Address
Why	Event or transaction of interest to the enterprise. These events keep the business afloat. That is, "*Why* is the business in business?"	Order, Return, Complaint, Withdrawal, Deposit, Compliment, Inquiry, Trade, Claim
How	Documentation of the event of interest to the enterprise. Documents record the events, such as a Purchase Order recording an Order event. That is, "*How* does the business stay in business?"	Invoice, Contract, Agreement, Account, Purchase Order, Speeding Ticket

Entity instances are the occurrences or values of a particular entity. Think of a spreadsheet as being an entity, with the column headings representing the pieces of information about the entity. Each spreadsheet row containing the actual values represents an entity instance. The entity Customer may have multiple customer instances with names Bob, Joe, Jane, and so forth. The entity Account can have instances of Bob's checking account, Bob's savings account, Joe's brokerage account, and so on.

Entity Types

The beauty of data modeling is that you can take the same information and show it at different levels of detail depending on the audience. The previous chapter introduced the three levels of detail: subject area, logical, and physical. Entities are components of all three levels.

For an entity to exist at a subject area level, it must be both basic and critical to the business. What is basic and critical depends very much on the concept of scope. At a universal level, there are certain subject areas common to all companies, such as Customer, Product, and Employee. Making the scope slightly narrower, a given industry may have certain unique subject areas.

Phone Number, for example, will be a valid subject area for a telecommunications company, but perhaps not for other industries, such as manufacturing. Each company may have subject areas that are unique to its business or its way of doing business. For example, Complaint could be a subject area for a consumer affairs department. Person and Company could be valid subject areas in our business card example.

Entities at a logical level represent the business at a more detailed level than at the subject area level. In general, a subject area entity represents many logical model entities. Examining the subject area Address in more detail could produce a large number of logical entities, including Email Address, Web Address, and Mailing Address.

At a physical level, the entities correspond to database tables. The rigor applied to the logical model is reversed, at times, to make applications perform well or to manage space more efficiently. Web Address and Email Address could be logical entities that translate directly into physical tables. However, if there is a reporting requirement to view all virtual address information, we may decide to combine both Web Address and Email Address into the same physical entity. With very large volumes of data, we might also decide to break up Email Address into several physical entities, each of a more manageable size. So at times, one logical entity can break down into several physical tables, and even more frequently, one physical table can be created from many logical entities.

An entity is shown as a rectangle with its name inside. Figure 4.1 contains several entities from our gelato store.

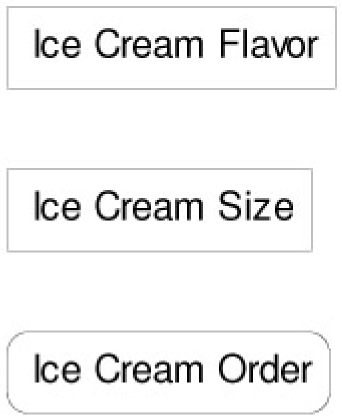


Figure 4.1: Sample entities

Notice that there are two types of rectangles: those with straight corners, such as Ice Cream Flavor and Ice Cream Size, and those with rounded edges, such as Ice Cream Order. Without introducing archaic data modeling jargon, it is enough to know that in most tools, the rectangles with straight right angle corners are strong and those with rounded corners are weak.

Strong entities stand on their own. They represent one occurrence of a person, place or thing independent of any other entities. In order to find the information about a particular Customer, for example, its **Customer Identifier** could be used to retrieve it from the database. "This is

Bob, Customer Identifier 123." An Ice Cream Flavor of 'Chocolate' might be retrieved with 'C'. An Ice Cream Size of '2 Scoops' might be retrieved with simply the number '2'.

Weak entities need to rely on at least one other entity. This means you *cannot* retrieve an entity instance without referring to an entity instance from another entity. For example, Ice Cream Order, might be retrieved by an Ice Cream Flavor or Ice Cream Size, *in combination with* something within Ice Cream Order such as a **Sequence Number**.

A data model is a communication tool. Distinguishing strong from weak entities on the model helps us understand the relationships and dependencies between entities. For example, a developer reading a data model showing that Ice Cream Order is a weak entity that depends on Ice Cream Flavor, would develop the application program to ensure that an ice cream flavor is present before orders for it are placed. That is, 'Chocolate' must be available as a flavor in the software system before an order for chocolate ice cream may be placed.

Exercise 4: Defining Subject Areas

List three subject areas in your organization. Does your organization have a single, agreed-upon definition for each of these subject areas? If not, why not? Can you achieve a single definition for each subject area?

Key Points

- An entity represents a collection of information about something that the business deems important and worthy of capture. An entity fits into one of several categories - who, what, when, where, why, or how.
- A noun or noun phrase identifies a specific entity.
- Entity instances are the occurrences or values of a particular entity.
- An entity can exist at the subject area, logical, or physical level of detail.
- An entity can be strong or weak.

Chapter 5: What are Data Elements?

Overview

Spreadsheets have columns
Just like data elements
Models all around

This chapter defines the concept of a data element and the three different levels at which a data element can exist: subject area, logical, and physical. Domains and the different types of domains are also discussed.

Data Element Explained

A data element is a property of importance to the business whose values contribute to identifying, describing, or measuring instances of an entity. The data element **Claim Number** identifies each claim. The data element **Student Last Name** describes the last name of each student. The data element **Gross Sales Amount** measures the monetary value of a transaction.

Returning to our spreadsheet analogy, the column headings on a spreadsheet are data elements. The cells beneath each column heading are the values for that column heading. Data elements can be thought of as the column headings in a spreadsheet, the fields on a form, or the labels on a report.

Ice Cream Flavor Name and **Ice Cream Size** are examples of data elements from our gelato store. **Company Name** and **Phone Number** are examples from the business card example

Data Element Types

As with entities, data elements can exist at subject area, logical, and physical levels. A data element at the subject area level must be a concept both basic and critical to the business. We do not usually think of data elements as subject areas, but depending on the business need, they can be. When I worked for a telecommunications company, **Telephone Number** was a data element that was so important to the business that it was represented on a number of subject area models.

A data element on a logical data model represents a business property. Each data element shown contributes to the business solution and is independent of any technology, including software and hardware. For example, **Ice Cream Flavor Name** is a logical data element because it has business significance regardless of whether records are kept in a paper file or within the fastest database out there.

A data element on a physical data model represents a database column. The logical data element **Ice Cream Flavor Name** might be represented as the physical data element ICE_CRM_FLVR_NAM.

I use the term *data element* throughout the text for consistency. However, I would recommend using the term that is most comfortable for your audience. For example, a business analyst might prefer the term 'attribute' or 'label', while a database administrator might prefer the term 'column' or 'field'.

Domain Explained

The complete set of all possible values that a data element may have is called a domain. A domain is a set of validation criteria that can be applied to more than one data element. For example, the domain 'Date', which contains all possible valid dates, can be assigned to any of these data elements:

- Employee Hire Date
- Order Entry Date
- Claim Submit Date
- Course Start Date

A data element may never contain values outside of its assigned domain. The domain values are defined by specifying the actual list of values or a set of rules. **Employee Gender Code**, for example, may be limited to the domain of (*female, male*).

Employee Hire Date may initially be assigned the rule that its domain contain only valid dates, for example. Therefore, this may include values such as

- February 15th, 2005
- 25 January 1910
- 20030410
- March 10th, 2050

Because **Employee Hire Date** is limited to valid dates, it does not include February 30th, for example. We can restrict a domain with additional rules. For example, by restricting the **Employee Hire Date** domain to dates earlier than today's date, we would eliminate March 10th, 2050. By restricting **Employee Hire Date** to YYYYMMDD (that is, year, month, and day concatenated), we would eliminate all the examples given except for 20030410. Another way of refining this set of values is to restrict the domain of **Employee Hire Date** to dates that fall on a Monday, Tuesday, Wednesday, Thursday, or Friday (that is, the typical workweek).

In our example of the business card, **Contact Name** may contain thousands or millions of values.

The values from our four sample cards in Figure 1.2 would be

- Steve Hoberman
- Steve
- Jenn
- Bill Smith
- Jon Smith

This name domain may need a bit of refining. It may be necessary to clarify whether a valid domain value is composed of both a first and last name, such as 'Steve Hoberman', or just a first name, such as 'Steve'. Could this domain contain company names such as 'IBM', as well? Could this domain contain numbers instead of just letters, such as the name R2D2 from the movie Star Wars? Could this domain contain special characters, such as the name O(+>? O(+>, representing

"The Artist Formerly Known as Prince" (the musician Prince changed his name to this unpronounceable "Love Symbol" in 1993).

There are three different types of domains:

- **Format.** Format domains specify the standard types of data one can have in a database. For example, Integer, Character(30), and Date are all format domains. The format domain for Ice Cream Size might be Character(15), meaning a particular Ice Cream Size can contain any sequence of characters and be at most 15 characters in length.
- **List.** List domains are similar to a drop-down list. They contain a finite set of values from which to choose. List domains are refinements of format domains. The format domain for **Order Status Code** might be Character(10). This domain can be further defined through a list domain of possible values {Open, Shipped, Closed, Returned}. The list domain for Ice Cream Size would be {one scoop, two scoops, three scoops}.
- **Range.** Range domains allow all values that are between a minimum and maximum value. For example, **Order Delivery Date** must be between Today's Date and three months in the future. As with list domains, range domains are a refined version of a format domain.

Domains are very useful for a number of reasons:

- **Improves data quality by checking against a domain before inserting data.** This is the primary reason for having a domain. By limiting the possible values of a data element, the chances of bad data getting into the database are reduced. For example, if every data element that represents money is assigned the Amount domain, consisting of all decimal numbers up to 15 digits in length including two digits after the decimal point, then there is a good chance that each of these data elements actually do contain currency. **Gross Sales Amount**, which is assigned the amount domain, would not allow the value 'R2D2' to be added.
- **The data model communicates even more.** When we display domains on a data model, the data model communicates that a particular data element has the properties of a particular domain and therefore, the data model becomes a more comprehensive communication tool. We learn, for example, that **Gross Sales Amount, Net Sales Amount**, and **List Price Amount** all share the Amount domain and therefore, share properties such that their valid values are limited to currency.
- **Greater efficiency in building new models and maintaining existing models.** When a data modeler embarks on a project, she can use a standard set of domains, thereby saving time by not reinventing the wheel. Any new data element that ends in Amount, for example, would be associated with the standard Amount domain, saving analysis and design time.

Exercise 5: Assigning Domains

1. What is the most appropriate domain for each of the three data elements below?

- Email Address

- Gross Sales Amount
- Country Code

Answers

1. Here are the domains for each of the following three data elements.

EMAIL ADDRESS

Based upon information from Wikipedia:

- *An e-mail address is a string of a subset of characters separated into 2 parts by an "@", a "local-part" and a domain, that is, local-part@domain. The local-part of an e-mail address may be up to 64 characters long and the domain name may have a maximum of 255 characters. However, the maximum length of the entire e-mail address is 254 characters.*

The local-part of the e-mail address may use any of these characters:

- *Uppercase and lowercase English letters (a–z, A–Z)*
- *Digits 0 through 9*
- *Characters ! # $ % & ' * + - / = ? ^ _ ` { | } ~*
- *Character . (dot, period, full stop) provided that it is not the first or last character, and provided also that it does not appear two or more times consecutively.*
- *Additionally, quoted-strings (i.e.: "John Doe"@example.com) are permitted, thus allowing characters that would otherwise be prohibited, however they do not appear in common practice.*

GROSS SALES AMOUNT

A format domain of Decimal(15,4). Both negative and positive numbers are acceptable.

COUNTRY CODE

As part of the ISO 3166-1993 standard, Country Code is two characters in length, and is a list domain consisting of over 200 values. Here is a partial list:

Open table as spreadsheet

Code	Definition and Explanation
AD	Andorra
AE	United Arab Emirates
AF	Afghanistan

AG	Antigua & Barbuda
AI	Anguilla
AL	Albania
AM	Armenia
AN	Netherlands Antilles
AO	Angola
AQ	Antarctica
AR	Argentina
AS	American Samoa
AT	Austria
AU	Australia
AW	Aruba
AZ	Azerbaijan
ZM	Zambia
ZR	Zaire
ZW	Zimbabwe
ZZ	Unknown or unspecified country

Just the codes beginning with 'A' or 'Z' are shown. 'ZZ' is an interesting country, and illustrates how easy it is to circumvent a business rule. That is, if we don't know the country and Country Code is required, we can always assign a 'ZZ' for 'Unknown'.

Key Points

- A data element is a property of importance to the business whose values contribute to identifying, describing or measuring instances of an entity.
- A domain is a set of validation criteria that can be applied to more than one data element.
- There are different types of domains, including format, list, and range domains.

- # Chapter 6: What are Relationships?
- **Overview**
- *Rules all around us*
  Relationships tell the tale
  Connecting the dots
- This chapter defines rules and relationships and the three different levels at which relationships can exist: subject area, logical, and physical. Data rules are distinguished from action rules. Cardinality and labels are explained so that you can read any data model as easily as reading a book. Other types of relationships, such as recursive relationships and subtyping, are discussed, as well.

Relationship Explained

In its most general sense, a rule is an instruction about how to behave in a specific situation. The following are examples of rules that you are familiar with:

- Your room must be cleaned before you can go outside and play.
- If you get three strikes, you are out and it is the next batter's turn.
- The speed limit is 55 mph.

Rules are visually captured on our data model through relationships. A relationship is displayed as a line connecting two entities. It captures the rules between these two entities. If the two entities are Employee and Department, the relationship may capture the rules "Each Employee must work for one Department" and "Each Department may contain many Employees."

Relationship Types

A rule can be either a data rule or an action rule. Data rules are instructions on *how* data relate to one another. Action rules are instructions on *what* to do when data elements contain certain values. Let's talk about data rules first.

There are two types of data rules - structural and referential integrity (RI) data rules. Structural rules (also known as cardinality rules) define the quantity of each entity instance that can participate in a relationship. For example:

- Each product can appear on one or many order lines.
- Each order line must contain one and only one product.
- Each student must have a unique student number.

RI rules focus on ensuring valid values:

- An order line cannot exist without a valid product.
- A claim cannot exist without a valid policy.
- A student cannot exist without a valid student number.

When we define a structural rule, we get the corresponding RI rule for free. For example, if we define this structural rule on our data model, "Each order line must contain one and only one product", it is automatically assumed and included that "An order line cannot exist without a valid product."

Action rules on the other hand, are instructions on *what to do* when data elements contain certain values:

- Freshman students can register for at most 18 credits a semester.
- A policy must have at least three claims against it to be considered high-risk.
- Take 10% off of an order if the order contains more than five products.

In our data models, we can represent the data and enforce data rules, but we cannot enforce action rules on a data model. A student data model can capture the level of student, such as Freshman or Senior, as well as the number of credits each student is taking each semester, but cannot enforce that a freshman student register for no more than 18 credits a semester.

Returning to our ice cream example, I eventually ordered a double scoop of gelato in a cone - one scoop of Chocolate and one scoop of Banana. Many relationships can describe the process of placing this order, such as:

- An ice cream container can be either a cone or cup.
- Each ice cream container can contain many scoops of ice cream.
- Each ice cream scoop must reside in an ice cream container (or our hands would get really sticky holding that scoop of banana gelato).
- Each ice cream flavor can be chosen for one or many ice cream containers.
- Each ice cream container can contain many flavors.

The three levels of granularity that apply to entities and data elements, also apply to the relationships that connect entities. Subject area relationships are high level rules that connect key concepts. Logical relationships are more specific and enforce the rules between the logical entities. Physical relationships are also specific rules and apply to the physical entities that the relationship connects. These physical relationships eventually become database constraints, which ensure that data adheres to the rules. So, in our ice cream example, "Each ice cream container can contain many scoops of ice cream", can be a subject area relationship. This high-level rule can be broken down into more detailed, logical relationships, such as defining the rule on the different types of containers: "An ice cream container can be either a cone or cup." This logical relationship then translates into the physical relationship "An ice cream container must be of one ice cream container type, whose values are 'cone' or 'cup' or 'not applicable'.

Cardinality Explained

Cardinality defines the number of instances of each entity that can participate in a relationship. It is represented by the symbols that appear on both ends of a relationship line. It is through cardinality that the data rules are specified and enforced. Without cardinality, the most we can say about a relationship is that two entities are connected in some way through a rule. For example, Person and Company have some kind of relationship, but we don't know much more than this.

The domain of values to choose from to represent cardinality on a relationship is limited to three values: zero, one, or many. *Many* (some people read it as *more*), means any number greater than one. Each side of a relationship can have any combination of zero, one, or many. Specifying zero or one allows us to capture whether or not an entity instance is required in a relationship. Specifying one or many allows us to capture "how many" of a particular instance participates in a given relationship.

Because we have only three cardinality symbols, we can't specify an exact number (other than through documentation), as in "A car has four tires." We can only say, "A car has many tires." A data model represents something in the real world. In capturing this something, there is always a tradeoff between refinement and simplicity. The greater the variety of symbols we show on a model, the more we can communicate. But more symbols also means greater complexity. Data modeling (using the notation in this book), forfeits a certain amount of refinement for simplicity. The advantage is we can explain very complex ideas with a simple set of symbols. In fact, I taught my six-year-old cardinality and she can now read the business rules on a data model. She is the only person in her Kindergarten class who can read cardinality (I am so proud!).

Each of the cardinality symbols is illustrated in the following example of Ice Cream Flavor and Ice Cream Scoop. An ice cream flavor is a selection choice for an ice cream scoop. An ice cream scoop must be one of the available ice cream flavors. Formalizing the rules between flavor and scoop, we have:

- Each Ice Cream Flavor can be the selection choice for one or many Ice Cream Scoops.
- Each Ice Cream Scoop must contain one Ice Cream Flavor.

Figure 6.1 captures these business rules.

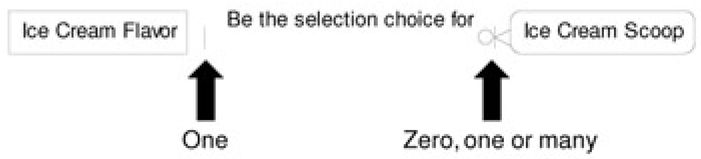


Figure 6.1: Ice Cream Flavor and Ice Cream Scoop, take 1

The small line means "one." The circle means "zero." The triangle with a line through the middle means "many." Some people call the "many" symbol a *crow's foot*. Relationship lines are frequently labeled to clarify the relationship and express the rule that the relationship represents. Thus, the label "Be the selection choice for" on the line in this example, helps in reading the relationship and understanding the rule.

Having a zero in the cardinality means we can use optional-sounding words such as 'may' or 'can' when reading the relationship. Without the zero, we use mandatory-sounding terms such as 'must' or 'have to'.

So instead of being redundant and saying:

- Each Ice Cream Flavor can be the selection choice for *zero*, one or many Ice Cream Scoops.

We take out the word 'zero' because it can be expressed using the word 'can', which implies the zero:

- Each Ice Cream Flavor can be the selection choice for one or many Ice Cream Scoops.

A relationship has a parent and child. The parent entity appears on the "one" side of the relationship, and the child appears on the "many" side of the relationship. When I read a relationship, I always start with the entity on the one side of the relationship first. "Each Ice Cream Flavor can be the selection choice for one or many Ice Cream Scoops." It's then followed by reading the relationship from the many side: "Each Ice Cream Scoop must contain one Ice Cream Flavor." In truth, it doesn't matter which side you start from, as long as you are consistent.

I also always use the word 'each' in reading a relationship, starting with the parent side. The reason for the word 'each' is that you want to specify, on average how many instances of one entity relate to a different entity instance. 'Each' is a more user-friendly term to me than 'A'.

Let's change the cardinality slightly and see how this impacts the resulting business rule. Assume that because of the rough economy, this ice cream shop decides to allow consumers to select more than one flavor in a scoop. Figure 6.2 contains the updated cardinality.

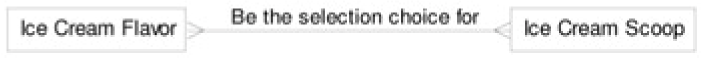


Figure 6.2: Ice Cream Flavor and Ice Cream Scoop, take 2

This is known as a many-to-many relationship, in contrast to the previous example, which was a one-to-many relationship. The business rules here are read as follows:

- Each Ice Cream Flavor must be the selection choice for many Ice Cream Scoops.
- Each Ice Cream Scoop must contain many Ice Cream Flavors.

Make sure the labels on relationship lines are as descriptive as possible. Here are some examples of good label names:

- contain
- work for
- own
- initiate
- categorize
- apply to

Always avoid the following words as label names, as they provide no additional information to the reader (you can use these words in combination with other words to make a meaningful label name; just avoid using these words by themselves):

- has
- have
- associate
- participate
- relate
- be

For example, replace the relationship sentence:

- A Person is *associated* with one Company.

With:

- A Person is *employed* by one Company.

Many modelers capture labels on both sides of the relationship line, instead of just one side, as shown in this chapter. In weighing simplicity versus precision, I chose simplicity. The other label can be inferred from the label that appears on the model. For example, I assumed the label 'contain' in Figure 6.1 and read the rule from Ice Cream Scoop to Ice Cream Flavor this way: "Each Ice Cream Scoop must contain one Ice Cream Flavor."

Recursion Explained

A recursive relationship is a rule that exists between instances of the same entity. A one-to-many recursive relationship describes a hierarchy, whereas a many-to-many relationship describes a network. In a hierarchy, an entity instance has at most one parent. In a network, an entity instance can have more than one parent. Let's illustrate both types of recursive relationships using Employee. See Figure 6.3 for a one-to-many recursive example and Figure 6.4 for a many-to-many example.

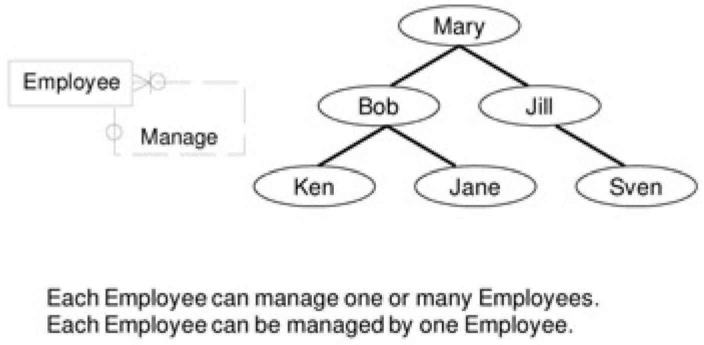

Employee
Manage
Mary
Bob
Jill
Ken
Jane
Sven
Each Employee can manage one or many Employees.
Each Employee can be managed by one Employee.

Figure 6.3: An Employee can work for one Manager

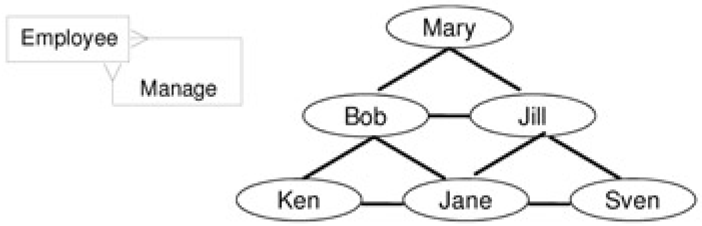


Figure 6.4: An Employee must work for many Managers

Using sample values such as 'Bob' and 'Jill' and sketching a hierarchy or network can really help understand, and therefore validate, cardinality. In Figure 6.3, for example, where the one-to-many captures a hierarchy, each employee has at most one manager. Yet in Figure 6.4 where the many-to-many captures a network, each employee must have many managers, such as Jane working for Bob, Jill, Ken, and Sven. (I would definitely update my resume if I were Jane.)

It is interesting to note that in Figure 6.3, there is optionality on both sides of the relationship. In this example, it implies we can have an Employee who has no boss (such as Mary) and an Employee who is not a manager (such as Ken, Jane, and Sven).

Data modelers have a love-hate relationship with recursion. On the one hand, recursion makes modeling a complex business idea very easy and leads to a very flexible modeling structure. We can have any number of levels in an organization hierarchy in Figure 6.3, for example. On the other hand, some consider using recursion to be taking the easy way out of a difficult modeling situation. There are many rules that can be obscured by recursion. For example, where is the Regional Management Level in Figure 6.4? It is hidden somewhere in the recursive relationship. Those in favor of recursion argue that you may not be aware of all the rules and that recursion protects you from having an incomplete model. The recursion adds a level of flexibility that ensures that any rules not previously considered are also handled by the model. It is therefore wise to consider recursion on a case-by-case basis, weighing obscurity against flexibility.

Subtyping Explained

Subtyping groups the common data elements and relationships of entities, while retaining what is unique within each entity. Subtyping is an excellent way of communicating that certain concepts are very similar.

In our ice cream example, we are told that an ice cream cone and ice cream cup can each contain many scoops of ice cream, as illustrated in Figure 6.5.

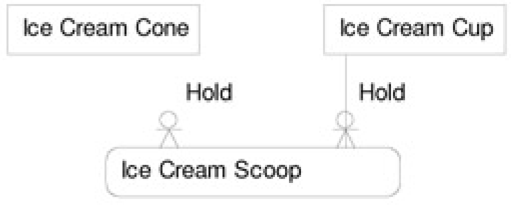


Figure 6.5: Ice cream example before subtyping

- Each Ice Cream Cone can hold one or many Ice Cream Scoops.
- Each Ice Cream Scoop must be held in one Ice Cream Cone.
- Each Ice Cream Cup can hold one or many Ice Cream Scoops.
- Each Ice Cream Scoop must be held in one Ice Cream Cup.

Rather than repeat the relationship to Ice Cream Scoop twice, we can introduce subtyping, as shown in Figure 6.6.

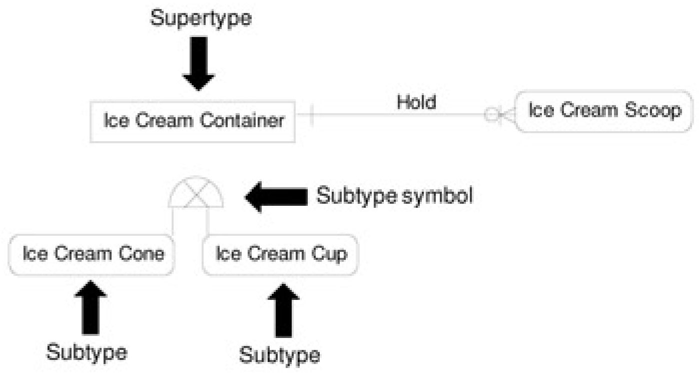


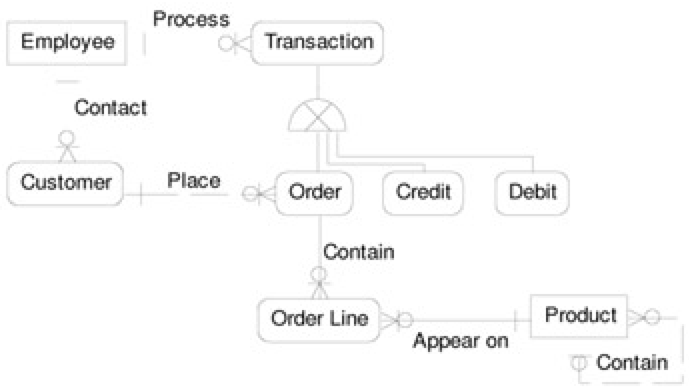
Figure 6.6: Ice cream example after subtyping

- Each Ice Cream Container can hold one or many Ice Cream Scoops.
- Each Ice Cream Scoop must be held in one Ice Cream Container.
- Each Ice Cream Container can be either an Ice Cream Cone or an Ice Cream Scoop.
- Each Ice Cream Cone is an Ice Cream Container.
- Each Ice Cream Cup is an Ice Cream Container.

The subtyping relationship implies that all of the properties from the supertype are inherited by the subtype. Therefore, there is an implied relationship from Ice Cream Cone to Ice Cream Scoop as well as Ice Cream Cup to Ice Cream Scoop. Not only does subtyping reduce redundancy on a data model, it makes it easier to communicate similarities across what otherwise would appear to be distinct and separate concepts.

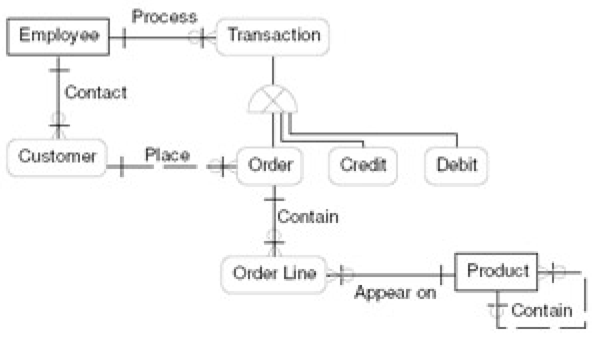
Exercise 6: Reading a Model

1. Practice reading the relationships in this model. See the Appendix for my answers. ?



Answers

1. Recall the model:



Each Employee may process one or many Transactions.

Each Transaction must be processed by one Employee.

Each Transaction may be either an Order, Credit, or Debit.

Each Order is a Transaction.

Each Credit is a Transaction.

Each Debit is a Transaction.

Each Employee may contact one or many Customers.

Each Customer must be contacted by one Employee.

Each Customer may place one or many Orders.

Each Order must be placed by one Customer.

Each Order may contain one or many Order Lines.

Each Order Line must belong to one Order.

Each Product may appear on one or many Order Lines.

Each Order Line must reference one Product.

Each Product may contain one or many Products.

Each Product may belong to one Product.

Key Points

- A rule is visually captured on a data model by a line connecting two entities, called a relationship.
- Data rules are instructions on *how* data relate to one another. Action rules are instructions on *what to do* when data elements contain certain values.
- Cardinality is represented by the symbols on both ends of a relationship that define the number of instances of each entity that can participate in the relationship. The three simple choices are zero, one, or many.
- Labels are the verbs that appear on the relationship lines. Labels should be as descriptive as possible to retain data model precision.
- A recursive relationship is a rule that exists between instances of the same entity.
- Subtyping groups the common properties of entities while retaining what is unique within each entity.

Chapter 7: What are Keys?

Overview

More than one John Doe
Which is the right Customer?
Recall by the key

There is a lot of data out there, but how do you sift through it all to find what you're looking for? That's where keys come in. Keys allow us to efficiently retrieve data, as well as navigate from one physical table to another. This chapter defines keys and distinguishes between the terms candidate, primary, and alternate keys. Surrogate keys and foreign keys and their importance are also explained.

Key Explained

Data elements identify, describe, or measure the entity instances in which they reside. There is often a need to find specific entity instances using one or more data elements. Those data element(s) that allow us to find specific entity instances are known as keys. The Library of Congress assigns an ISBN (International Standard Book Number) to every book. When the ISBN for this book, 9780977140060, is entered into many search engines and database systems, the book entity instance **Data Modeling Made Simple** will be returned (try it!). A particular tax identifier can help us find an organization. The key **Account Code** can help us find a particular account.

Candidate Key Explained

A candidate key is one or more data elements that uniquely identify an entity instance. Sometimes a single data element identifies an entity instance, such as ISBN for a book, or **Account Code** for an account. Sometimes it takes more than one data element to uniquely identify an entity instance. For example, both a **Promotion Code** and **Promotion Start Date** are necessary to identify a promotion. When more than one data element makes up a key, we use the term 'composite key'. So **Promotion Code** and **Promotion Start Date** together are a composite candidate key for a promotion.

A candidate key has three main characteristics:

- **Unique.** There cannot be duplicate values in the data in a candidate key and it cannot be empty (also known as 'nullable'). Therefore, the number of distinct values of a candidate key must be equal to the number of distinct entity instances. If the entity Book has ISBN as its candidate key, and if there are 500 book instances, there will also be 500 unique ISBNs.
- **Non-volatile.** A candidate key value on an entity instance should never change. Since a candidate key is used to find a unique entity instance, you would be unable to find that instance if you were still trying to use the value before it was changed. Changing a candidate key would also mean changing it in every other entity in which it appears with the original value.
- **Minimal.** A candidate key should contain only those data elements that are needed to uniquely identify an entity instance. If four data elements are listed as the composite candidate key for an entity, but only three are really needed for uniqueness, then only those three should make up the candidate key.

Figure 7.1 contains a data model before candidate keys have been identified.

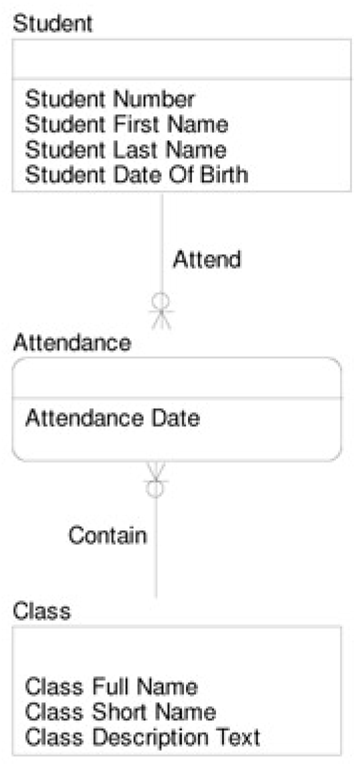


Figure 7.1: Data model before candidate keys have been identified

- Each Student may attend one or many Classes.
- Each Class may contain one or many Students.

Note that we have a many-to-many relationship between Student and Class that was replaced by the entity Attendance and two one-to-many relationships (more on this in our normalization section). In reading a many-to-many relationship, I have found it helpful to ignore the entity in the middle (Attendance, in this example) and just read the labels between the entities on either side. For example, each Student may attend one or many Classes and each Class may contain one or many Students.

Table 7.1 contains sample values for each of these entities.

Table 7.1: Sample values for Figure 7.1 ➡Open table as spreadsheet

Student			
Student Number	**Student First Name**	**Student Last Name**	**Student Date Of Birth**
SM385932	Steve	Martin	1/25/1958
EM584926	Eddie	Murphy	3/15/1971
HW742615	Henry	Winkler	2/14/1984
MM481526	Mickey	Mouse	5/10/1982

Student			
Student Number	**Student First Name**	**Student Last Name**	**Student Date Of Birth**
DD857111	Donald	Duck	5/10/1982
MM573483	Minnie	Mouse	4/1/1986
LR731511	Lone	Ranger	10/21/1949
EM876253	Eddie	Murphy	7/1/1992

➡Open table as spreadsheet

Attendance			
Attendance Date			
5/10/2009			
6/10/2009			
7/10/2009			

➡Open table as spreadsheet

Class			
Class Full Name	**Class Short Name**	**Class Description Text**	
Data Modeling Fundamentals	Data Modeling 101	An introductory class covering basic data modeling concepts and principles.	
Advanced Data Modeling	Data Modeling 301	A fast-paced class covering techniques such as advanced normalization and ragged hierarchies.	
Tennis Basics	Tennis One	For those new to the game of tennis, learn the key aspects of the game.	
Juggling		Learn how to keep three balls in the air at once!	

Based on our definition of a candidate key and a candidate key's characteristics of being unique, non-volatile, and minimal, what would you choose as the candidate keys for each of these entities?

For Student, **Student Number** appears to be a valid candidate key. There are eight students and eight distinct values for **Student Number**. So unlike **Student First Name** and **Student Last Name**, which can contain duplicates like Eddie Murphy, **Student Number** appears to be unique. **Student Date Of Birth** can also contain duplicates such as '5/10/1982' which is the Student Date

Of Birth for both Mickey Mouse and Donald Duck. However, the combination of **Student First Name**, **Student Last Name**, and **Student Date Of Birth** may make a valid candidate key.

For Attendance, we are currently missing a candidate key. Although the **Attendance Date** is unique in our sample data, we will probably need to know which student attended which class on this particular date.

For Class, on first glance it appears that any of its data elements are unique and would therefore qualify as a candidate key. However, Juggling does not have a **Class Short Name**. So because **Class Short Name** can be empty, we cannot consider it a candidate key. Also, one of the characteristics of a candidate key is that it is non-volatile. I know, based on my teaching experience, that class descriptions can change. Therefore, **Class Description Text** also needs to be ruled out as a candidate key, leaving **Class Full Name** as the best option for a candidate key.

Primary and Alternate Keys Explained

Even though an entity may contain more than one candidate key, we can only select one candidate key to be the primary key for an entity. A primary key is a candidate key that has been chosen to be *the* unique identifier for an entity. An alternate key is a candidate key that although unique, was not chosen as the primary key, but still can be used to find specific entity instances.

We have only one candidate key in the Class entity, so **Class Full Name** becomes our primary key. We have to make a choice, however, in Student, because we have two candidate keys. Which Student candidate key would you choose as the primary key?

In selecting one candidate key over another as the primary key, consider succinctness and security. Succinctness means if there are several candidate keys, choose the one with the fewest data elements or shortest in length. In terms of security, it is possible that one or more data elements within a candidate key will contain sensitive data whose viewing should be restricted. We want to avoid having sensitive data in our entity's primary key because the primary key can propagate as a foreign key and therefore spread this sensitive data throughout our database.

Considering succinctness and security in our example, I would choose **Student Number** over the composite **Student First Name**, **Student Last Name**, and **Student Date Of Birth**. It is more succinct and contains less sensitive data.

Figure 7.2 shows our data model updated with primary and alternate keys. Primary keys are shown above the line in an entity box. Different modeling tools use different notations for alternate keys. I have chosen to show the composite alternate key in Student in italics. Note that if there had been more than one alternate key in the same entity, I would need to use a different format to distinguish the alternate keys from each other.

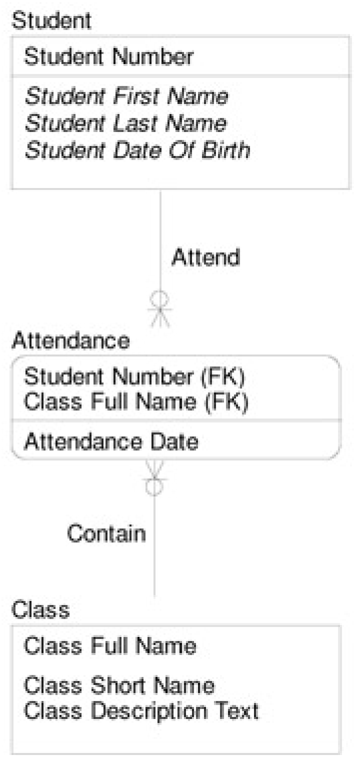


Figure 7.2: Data model updated with primary and alternate keys

Attendance now has as its primary key **Student Number** and **Class Full Name**, which appear to make a valid primary key. 'FK' stands for 'Foreign Key', which will be discussed shortly.

So to summarize, a candidate key consists of one or more data elements that uniquely identify an entity instance. The candidate key that is selected as the best way to identify each unique record in the entity becomes the primary key. The other candidate keys become alternate keys. Keys containing more than one data element are known as composite keys.

Surrogate Key Explained

A surrogate key is a primary key that substitutes for a natural key, which is what the business sees as the unique identifier for an entity. It has no embedded intelligence and is used by IT (and not the business) for integration or performance reasons.

Surrogate keys are useful for integration, which is an effort to create a single, consistent version of the data. Applications such as data warehouses often house data from more than one application or system. Surrogate keys enable us to bring together information about the same entity instance that is identified differently in each source system. If the same concept, such as Student or Class, exists in more than one system, there is a good chance some amount of integration will be necessary. For example, Robert Jones in system XYZ and Renee Jane in system ABC might both be identified in their respective systems as RJ. But if we tried to bring them together using RJ to link them, the data would be incorrect — we'd have Robert and Renee identified as the same person. Instead, a different, non-overlapping, surrogate key could be assigned to each of them. Similarly, if Robert Jones is identified as RJ in system XYZ and BJ in system DEF, the information about him could be consolidated under a single surrogate key value. The fact that they're the same would need to be determined through a separate effort.

Surrogate keys are also efficient. You've seen that a primary key may be composed of one or more attributes of the entity. A single surrogate key is more efficient to use than having to specify three or four (or five or six) attributes to locate the single record you're looking for.

When using a surrogate key, always make an effort to determine the natural key and then define an alternate key on this natural key. For example, assuming a surrogate key is a more efficient primary key than **Class Full Name**, we can create the surrogate key **Class Id** for Class and define an alternate key on **Class Full Name**, as shown in Figure 7.3. Table 7.2 contains the values in Class.

Table 7.2: Class values updated with surrogate key

➡Open table as spreadsheet

Class Id	Class Full Name	Class Short Name	Class Description Text
1	Data Modeling Fundamentals	Data Modeling 101	An introductory class covering basic data modeling concepts and principles.
2	Advanced Data Modeling	Data Modeling 301	A face-paced class covering techniques such as advanced normalization and ragged hierarchies.
3	Tennis Basics	Tennis One	For those new to the game of tennis, learn the key aspects of the game.
4	Juggling		Learn how to keep three balls in the air at once!

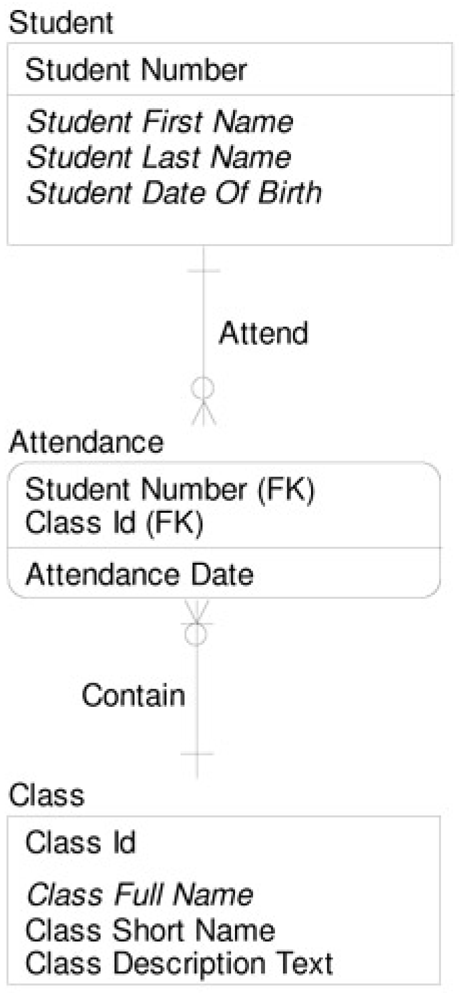


Figure 7.3: Data model updated with surrogate key

In this example, we're using **Class Full Name** as a candidate key, but in the real world, it's never a good idea to use a name as a key because names can change.

Foreign Key Explained

A foreign key is a data element that provides a link to another entity. A foreign key allows a database management system to navigate from one entity to another. For example, we need to know who owns an Account, so we would want to include the identifier of the customer to whom it belongs in the entity. The **Customer Id** in Account is the primary key of that Customer in the Customer entity. Using this foreign key back to Customer enables the database management system to navigate from a particular account or accounts, to the customer or customers that own each account. Likewise, the database can navigate from a particular customer or customers, to find all of their accounts.

A foreign key is automatically created when we define a relationship between two entities. When a relationship is created between two entities, the entity on the "many" side of the relationship inherits the primary key from the entity on the "one" side of the relationship.

In Figure 7.3, there are two foreign keys in the Attendance entity. The **Student Number** foreign key points back to a particular student in the Student entity. The **Class Id** foreign key points back to a particular Class in the Class entity. Table 7.3 contains a few Attendance entity instances.

Table 7.3: Attendance entity instances

Open table as spreadsheet

Student Number	Class Id	Attendance Date
SM385932	1	5/10/2009
EM584926	1	5/10/2009
EM584926	2	6/10/2009
MM481526	2	6/10/2009
MM573483	2	6/10/2009
LR731511	3	7/10/2009

By looking at these values and recalling the sample values from Tables 7.1 and 7.2, we learn that Steve Martin and Eddie Murphy both attended the Data Modeling Fundamentals class on 5/10/2009. Eddie Murphy also attended the Advanced Data Modeling Class with Mickey and Minnie Mouse on 6/10/2009. Lone Ranger took Tennis Basics (by himself as usual) on 7/10/2009.

Exercise 7: Clarifying Customer Id

1. I was showing examples of both complete and incomplete definitions during a recent training class, when I shared the following incomplete definition for a **Customer Id**:

    - *A **Customer Id** is the unique identifier for a Customer.*

    "What else can you say about **Customer Id** anyway?" a participant asked.

    What else can you say about **Customer Id** (or any identifier) to add more meaning to its definition?

Answers

1. There are three terms within this definition that require an explanation: 'unique', 'identifier', and 'Customer'.

    DOCUMENT UNIQUENESS PROPERTIES

    The term 'unique' is ambiguous and could easily be interpreted differently by readers of this definition. To maintain clarity and correctness, these questions should be answered within the definition:

- Are identifier values ever reused?
- What is the scope of uniqueness?
- How is the identifier validated?

DOCUMENT THE CHARACTERISTICS OF THE IDENTIFIER

We can describe the actual identifier in more detail including addressing these areas:

- **Purpose.** For example, perhaps the identifier is needed because there are multiple source systems for Customer data, each with their own Id. To enable a common set of data to be held about them, this identifier needed to be created to facilitate integration and guarantee uniqueness across all customers.
- **Business or surrogate key.** Document whether the identifier is meaningful to the business (i.e. the business or natural key) or whether it is a meaningless integer counter (i.e. the surrogate key).
- **Assignment.** Document how a new customer identifier is assigned. The party that is responsible for creating new identifiers should also be mentioned.

DEFINE THE CUSTOMER

Because definitions should stand on their own, we also can define customer within this definition. We can reference the subject area definition of customer.

See the Appendix for my answers.

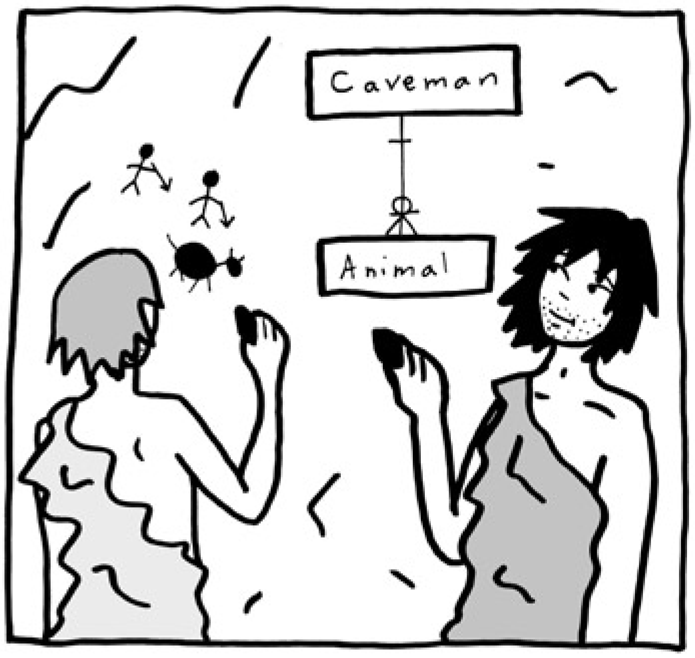
Key Points

- A key is a data element or set of data elements that helps us find entity instances.
- A candidate key is a set of one or more data elements that uniquely identify an entity instance.
- A candidate key becomes either a primary or alternate key.
- A primary key represents the one or more data elements that uniquely identify an instance of an entity and that is chosen to be *the* unique identifier everyone should use. In contrast, an alternate key also uniquely identifies entity occurrences, but is not chosen as *the* unique key.
- If a key contains more than one data element, it is known as a composite key.
- A surrogate key is a primary key with no embedded intelligence that is a substitute for a natural key. It is used by IT to facilitate integration and introduce database efficiencies.
- A foreign key points from one entity instance to another.

Section III: Subject Area, Logical, and Physical Data Models

Chapter List

Part Overview



Section III explores the three different levels of models: subject area, logical, and physical. You'll recall from the camera analogy that the format or film the camera uses is analogous to the three levels of modeling: subject area, logical, and physical. A subject area model (SAM) is analogous to a proof sheet that contains small thumbnail images of each photograph. It represents a business need. It is a very broad view, containing only the basic and critical concepts for a given scope. The logical data model (LDM) equates to the film negative; a perfect view of the picture, yet also impractical as you can't put a negative in a photo frame and show it to your friends. The LDM represents a detailed business solution, capturing the business requirements without complicating the model with implementation concerns such as software and hardware. The physical data model (PDM) equates to the actual photograph or slide or any instantiation of the negative, and represents a detailed technical solution. It loses some of the exactness of the LDM, but this loss usually comes with gains in performance and usability within a given hardware and software set.

In addition to these three levels of detail, there are also two different modeling mindsets: relational and dimensional. Relational modeling is the process of capturing how the business works, while dimensional modeling is the process of capturing what the business is monitoring or measuring. Relational modeling captures how the business works and captures business rules, such as "A Customer must have at least one Account", or a "Product must have a **Product Short Name**." Dimensional modeling captures what the business uses to measure how it is doing. For example, examining sales at a day level and then, after getting the answer, looking at sales at a month or year level, or a product or brand level, or a city or country level. Dimensional modeling is all about playing with numbers by summarizing or aggregating data such as sales amount.

The table on the facing page summarizes these three model levels and two mindsets, leading to six different types of models.

Open table as spreadsheet

		Mind set	
		Relational	**Dimensional**
Types of models	**SAM**	Key concepts and their business rules, such as a "Customer can place many Orders."	Key concepts focused around one or more measures, such as "I want to see Gross Sales Amount by Customer."
	LDM	All data elements required for a given application or business process, neatly organized into entities according to strict business rules and independent of technology, such as "Customer Last Name and Customer Shoe Size depend completely on Customer Identifier."	All data elements required for a given reporting application, focused on measures and independent of technology, such as "I want to see Gross Sales Amount by Customer and view the Customer's first and last name."
	PDM	The LDM modified for a specific technology, such as database or access software. For example, "To improve retrieval speed, we need a non-unique index on Customer Last Name."	The LDM modified for a specific technology, such as database or access software. For example, "Because there is a need to view Gross Sales Amount at a Day level, and then by Month and Year, we should consider combining all calendar data elements into a single table."

Each of these six models will be explained in detail in this section. Chapter 8 goes into detail on the subject area model, discussing the variations along with how to build this type of model. Chapter 9 focuses on the relational and dimensional logical data model. Chapter 10 focuses on the physical data model, going through the different techniques for building an effective design, such as denormalization and partitioning. Slowly Changing Dimensions (SCDs) are also discussed in this chapter.

Chapter 8: What are Subject Area Models?

Overview

Need the Big Picture?
No common definitions?
Hero, build a SAM!

The highlighted row in Table 8.1 shows the focus of this chapter - the subject area model (SAM).

Table 8.1: The SAM is the focus of this chapter

➡Open table as spreadsheet

	Relational	**Dimensional**
SAM	**'One-pager' on how something works**	**'One-pager' on what is monitored**
LDM	Detailed business solution on how something works	Detailed business solution on what is monitored
PDM	Detailed technical solution on how something works	Detailed technical solution on what is monitored

A subject area model shows the key concepts in a particular area and how these concepts interact with each other. This chapter defines a subject area followed by an explanation of the importance of the subject area model and subject area definitions. Then the three types of subject area models are discussed, including the application subject area model, where both relational and dimensional variations exist. I conclude this chapter with a summary of the ten-step approach to building a subject area model.

Subject Area Explained

A subject area is a key concept that is both *basic* and *critical* to your audience. "Basic" means this term is probably mentioned many times a day in conversations with the people who represent the audience for the model. "Critical" means the business would be very different or non-existent without this concept.

The majority of subject areas are easy to identify and include concepts that are common across industries, such as Consumer, Customer, Employee, and Product. An airline may call a Customer a Passenger, and a hospital may call a Customer a Patient, but they are still someone who purchases goods or services. Each subject area will be shown in much more detail at the logical and physical phases of design. For example, the Consumer subject area might encompass the logical entities Consumer, Consumer Association, Consumer Demographics, Consumer Type, and so on.

Many subject areas however, can be more challenging to identify as they may be subject areas to your audience but not to others in the same department, company, or industry. For example, a subject area named Account would most likely be a subject area for a bank and for a manufacturing company. However, the audience for the bank subject area model might also require Checking Account and Savings Account to be on their model, whereas the audience for the manufacturing subject area model might, instead, require General Ledger Account and Accounts Receivable Account to be on the model.

In our example with the business card, a basic and critical concept can be Address, but Mailing Address can also be basic and critical. Should the subject area model for contact management contain Mailing Address as well? To answer this question, we need to know whether Mailing Address is basic and critical to your audience. The key point about subject area modeling is to model at a level where the audience for the model would agree that each subject area is a key concept.

Subject Area Model Explained

A subject area model is a set of symbols and text representing the key concepts and rules binding these key concepts for a specific business or application scope, for a particular audience, that fits neatly on one page. It could be an 8 ½ x 11, 8 ½ x 14, or similar sized paper, but it cannot be a plotter-sized piece of paper. Limiting the subject area model to one page is important because it forces the modeler and participants to select only key concepts. On one page, for example, we can fit 10 or 20 or 50 concepts, but not 500 concepts. A good rule of thumb, therefore, is to ask yourself if the audience for this model would include this concept as one of the top 10 or 20 or 50 concepts in their business. This will rule out concepts that are at too low a level of detail. They will appear in the more detailed logical data model. If you're having trouble limiting the number of concepts, think about whether or not there are other concepts into which the ones you're discussing could be grouped. These higher concepts are the ones you should be including in the subject area model.

The subject area model includes subject areas, their definitions, and the relationships that show how these subject areas interact with each other. Unlike the logical and physical data models, as we will see, subject area models may contain many-to-many relationships. A sample subject area model appears in Figure 8.1.

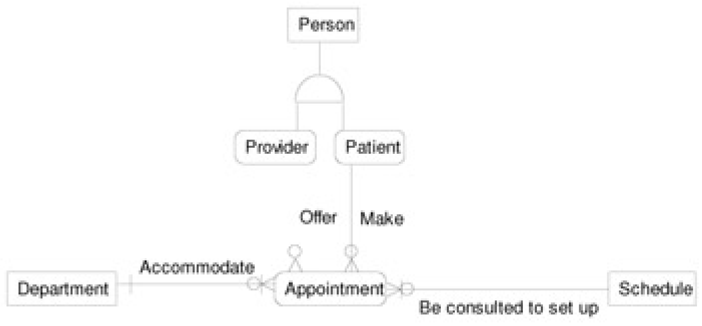


Figure 8.1: Healthcare Facility Appointment Subject Area Model

Subject area definitions:

Appointment	A meeting scheduled to take place at an assigned date and time.
Department	A specialized division of a large organization, such as the Accounting Department.
Patient	A person who requires medical care.
Person	A human being.
Provider	An individual who gives medical care.
Schedule	An ordered list of times at which events or activities are planned to occur.

Business Rules (listed in the order we would typically walk someone through the model):

- Each Person may be a Provider, a Patient, or both a Provider and Patient. Note that when the subtyping symbol does not have an 'X' in its center (as shown in Figure 8.1), it indicates that a member of the supertype can play more than one subtype role. This is called an inclusive (overlapping) subtype. For example, a particular person can be both a provider and a patient.
- Each Provider is a Person.
- Each Patient is a Person.
- Each Provider may offer one or many Appointments.
- Each Patient may make one or many Appointments.
- Each Schedule may be consulted to set up one or many Appointments.
- Each Department may accommodate one or many Appointments.
- Each Appointment must involve one Provider, one Patient, one Department, and one Schedule.

Notice on the model in Figure 8.1, that concepts such as Patient and Provider are likely to be considered subject areas throughout the healthcare industry. There are also slightly more detailed subject areas on this model, such as Schedule and Appointment, which are considered basic and critical and therefore subject areas for the particular audience for this subject area model. Yet these more detailed subject areas may not be considered subject areas within a different department in this same healthcare company, such as accounting and marketing.

During the subject area modeling phase, definitions of the concepts should be given a lot of attention. All too often, we wait until it is too late in the development process to get definitions. Waiting too long usually leads to not writing definitions altogether, or doing a rush job by writing quick definition phrases that have little or no usefulness. If the definitions behind the terms on a data model are nonexistent or poor, multiple interpretations of the subject area become a strong possibility. Imagine a business rule on our model that states that an employee must have at least one benefits package. If the definition of Employee is lacking, we may wonder, for example, whether this business rule includes job applicants and retired employees.

By agreeing on definitions at the concept level, the more detailed logical and physical analysis will go smoother and take less time. For example, definitions can address the question, "Does

Customer include potential customers or only existing customers?" See the cartoon in Figure 8.2 for an example of what not to do.

OK, we're almost done with user acceptance testing and everything looks great with this new marketing application. Just one small question – what is a Customer?

Figure 8.2: Clarify key concept definitions early!

We need to do a better job of capturing definitions. In fact, during a recent presentation to over 100 business analysts, I asked the innocent question, "How many of you have an agreed-upon single definition of Customer in your organization?" I was expecting at least a handful of the 100 participants to raise their hands, but no one in the room raised their hand!

There are several techniques for writing a good definition. One I like in particular is to write the definition as if you are explaining the term to a child. You wouldn't use many big words or restate the obvious. It also would not be too verbose, as a child may not have the same attention span as we do. Other techniques include avoiding defining a concept using only the terms in the name of the concept (e.g. "The Customer is our customer.") and naming a concept only after that concept's definition has already been written.

There are three main reasons why definitions are important:

- **Assists business and IT with decision making.** If a business user has a different interpretation of a concept than what was actually implemented, it is easy for poor decisions to be made, compromising the entire application. If a business user would like to know how many products were ordered each month, for example, imagine the poor judgments that could result if the user expected raw materials to be included with products and they were not. Or, what if he or she assumed that raw materials were not included, but they were?

- **Helps initiate, document and resolve different perspectives on the same concept.** The SAM is a great medium for resolving differences in opinion on the meaning of high-level terms. Folks in accounting and sales can both agree that Customer is an important concept. But can both groups agree on a common definition? If they can agree, you are one step closer to creating a holistic view of the business.
- **Supports data model precision.** A data model is precise. This assumes that the subject area definitions are also precise. An Order Line cannot exist without a Product, for example. However, if the definition of Product is missing or vague, we have less confidence in the concept and its relationships. Is a Product, for example, raw materials and intermediate goods, or only a finished item ready for resale? Can we have an order for a service, or must a product be a tangible deliverable? The definition is needed to support the Product concept and its relationship to Order Line.

When the subject area model is complete, which includes subject area definitions, it is a powerful tool that can provide a number of important business benefits:

- **Provides broad understanding.** We can capture extremely complex and encompassing business processes, application requirements, and even entire industries on a single piece of paper. This enables people with different backgrounds and roles to understand and communicate with each other on the same concepts, agreeing or debating on issues.
- **Defines scope and direction.** By visually showing subject areas and their business rules, we can more easily identify a subset of the model to analyze. For example, we can model the entire logistics department, and then scope out of this a particular logistics application that we would like to build. The broad perspective of a subject area model can help us determine how planned and existing applications will coexist. It can provide direction and guidance on what new functionality the business will need next.
- **Offers proactive analysis.** By developing a subject area-level understanding of the application, there is a strong chance we will be able to identify important issues or concerns, saving substantial time and money later on. Examples include subject area definition differences and different interpretations on project scope.
- **Builds rapport between IT and the business.** A majority of organizations have some level of internal communication issues between the business and IT departments. Building a subject area model together is a great way to remove or reduce these communication barriers. On one occasion, a key business user and I sketched out a Contact Data Mart subject area model, which built not just business understanding, but also a strong relationship with this key user.

Types of Subject Area Models

There are three types of subject area models. I've coined acronyms for them that are easy to remember: the Business Subject Area Model (BSAM), the Application Subject Area Model (ASAM), and the Comparison Subject Area Model (CSAM). The BSAM is a subject area model of a defined portion of the business, the ASAM is a subject area model of a defined portion of a particular application, and the CSAM is a subject area model that shows how something new fits within an existing environment.

Business Subject Area Model (BSAM)

The BSAM is a subject area model of a defined portion of the business (not an application). The scope can be limited to a department or function such as manufacturing or sales. It can be as broad as the entire company or industry. Company data models are often called enterprise data models. In addition to all the concepts within an organization, they can also contain external concepts such as government agencies, competitors, and suppliers. In addition to serving as a foundation for an enterprise architecture, a BSAM is also a very good place to start capturing the subject areas and business rules for a new application. All future subject area and logical data models can be based on this initial model.

Figure 8.3 contains an example of a BSAM. The purpose of this model was to understand the concepts and high-level business rules within the consumer interaction area in preparation for building a consumer interaction reporting application. A good first step is to understand how the consumer interaction business works. We could ask questions to learn more about what the business needs, such as:

- What is a Consumer? That is, can a Consumer be someone who has not yet purchased the Product?
- Can one Interaction be placed by more than one Consumer?
- Does an Interaction have to be for a specific Product or can an interaction be for something more general than a Product, such as a Brand?
- What types of Complaints do you receive?

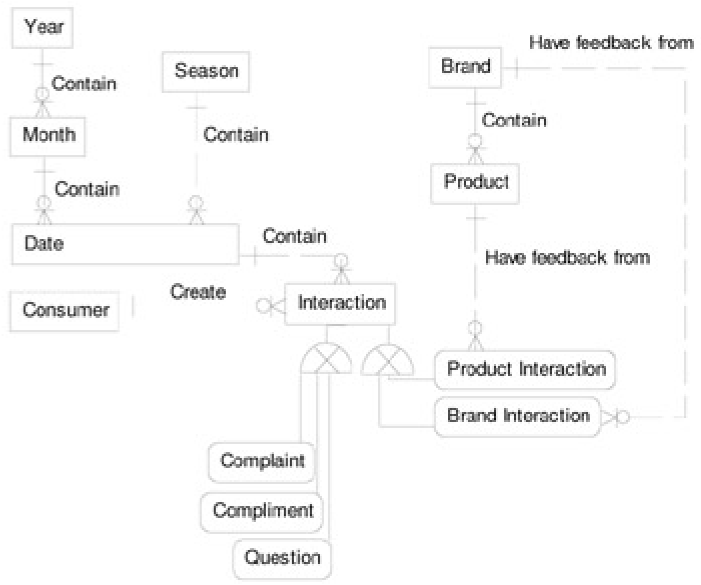


Figure 8.3: Consumer Interaction BSAM

Subject area definitions:

Brand A grouping of products that is recognized by consumers. A brand means something to a consumer in terms of the experience they will have if they

	purchase a product within this brand.
Brand Interaction	A contact initiated by one consumer about a specific brand.
Complaint	Negative feedback that we receive on one of our products or brands.
Compliment	Positive feedback that we receive on one of our products or brands.
Consumer	Someone who buys or receives products or services from us with the intent of the products or services being used and not resold; in other words, the final recipient of the product or service. For example, we are a consumer any time we purchase a product from one of our customers.
Date	A period of time containing 24 hours.
Interaction	A contact between an employee and a consumer for a specific product or brand. An interaction can take place through a variety of mediums, such as through phone, email, and mail. Interactions fit into one of three categories: complaints, compliments and questions. Here are examples of some of the interactions: • "I love your product." (compliment) • "I hate your product." (complaint) • "I found a strange object in your product." (complaint) • "Where can I buy your product?" (question) • "I found your product difficult to assemble." (complaint)
Month	One of the twelve major subdivisions of the year.
Product	The goods or services that are offered for sale.
Product Interaction	A contact initiated by one consumer about a specific product.
Question	An inquiry that we receive on one of our products or brands.
Season	A season is one of the major divisions of the year, generally based on yearly periodic changes in weather.
Year	A period of time containing 365 days (or 366 days in a leap year).

Business Rules (listed in the order we would typically walk someone through the model):

- Each Year may contain one or many Months. (Note that this is the way the business rule is read based on the relationship. The data modeling notation we are using is not so precise as to capture a rule such as "Each Year must contain 12 Months.")
- Each Month must belong to one Year.
- Each Month may contain one or many Dates.
- Each Date must belong to one Month.
- Each Season may contain one or many Dates.
- Each Date must belong to one Season.
- Each Brand may contain one or many Products.
- Each Product must belong to one Brand.
- Each Consumer may create one or many Interactions.

- Each Interaction must be created by one Consumer.
- Each Date may contain one or many Interactions.
- Each Interaction must occur on one Date.
- Each Interaction may be a Complaint, Compliment, or Question.
- Each Complaint is an Interaction.
- Each Compliment is an Interaction.
- Each Question is an Interaction.
- Each Interaction may be a Product Interaction or Brand Interaction.
- Each Product Interaction is an Interaction.
- Each Brand Interaction is an Interaction.
- Each Product may have feedback from one or many Product Interactions.
- Each Product Interaction must be feedback on one Product.
- Each Brand may have feedback from one or many Brand Interactions.
- Each Brand Interaction must be feedback on one Brand.

This model revealed a reporting challenge. One of the reporting requirements was to determine how many interactions are reported at a brand level. The model shows that interactions can be entered at a product level, which then role up to a brand. Or interactions can come in directly at the brand level. The challenge is to answer this question: "How many interactions are there at a brand level?" According to our data model, if we just summarize those product interactions up to a brand level (i.e., follow the relationship from Interaction to Product Interaction to Product to Brand), we are missing those interactions that occur only at the brand level. Therefore, we need to summarize product interactions up to a brand and then add those interactions at just a brand level to get the total number of interactions at a brand level to answer this question.

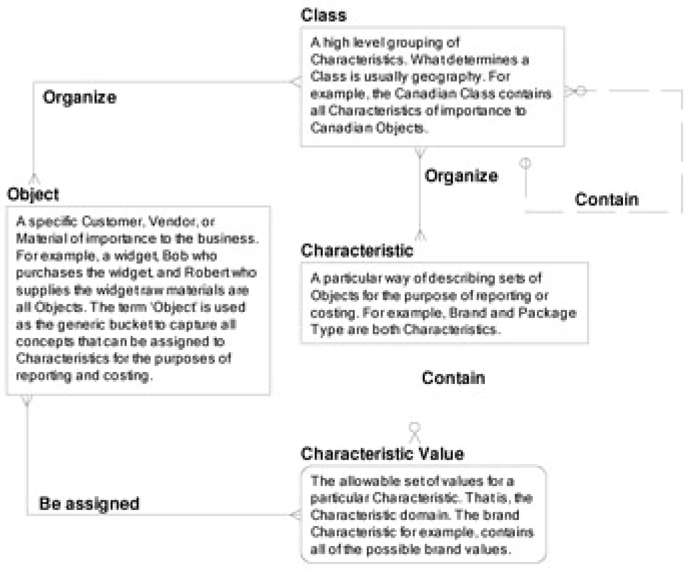
The BSAM is the most frequently built type of subject area model. Many times when we say we are creating a subject area model, we mean the BSAM. Before embarking on any large development effort, we first need to understand the business. If an organization needs a new claims-processing system, it needs to have a common understanding of claims and related subject areas. The BSAM can be created simply to understand a business area, or as the beginning of a large development effort, such as introducing packaged software into your organization.

Application Subject Area Model (ASAM)

The ASAM is a subject area model of a defined portion of a particular application. BSAMs are frequently the first step in large development efforts (first understand the business before you understand the application), so it is usually the starting point for the ASAM. The ASAM is often a subset of the BSAM. For example, after creating the BSAM of the human resources department, we can now carve out of it an ASAM for an application tracking employee training.

We mentioned earlier that there are two modeling mindsets: relational and dimensional. Relational and dimensional data models lead to relational- and dimensional-based applications, so we have two types of ASAMs: relational and dimensional. Relational ASAMs focus on how the business works through the eyes of the application, and dimensional ASAMs focus on how the business is monitored through the eyes of the application.

I once built a relational ASAM to capture the classifications concept in the Enterprise Resource Planning (ERP) software package, SAP/R3. In an effort to understand how SAP treats classifications (terminology, rules, and definitions), I created this model from studying screens, help files and a large quantity of the underlying 350 database tables within Classifications. Figure 8.4 contains a subset of this model.



A high level grouping of Characteristics. What determines a Class is usually geography. For example, the Canadian Class contains all Characteristics of importance to Canadian Objects.

A specific Customer, Vendor, or Material of importance to the business. For example, a widget, Bob who purchases the widget, and Robert who supplies the widget raw materials are all Objects. The term 'Object' is used as the generic bucket to capture all concepts that can be assigned to Characteristics for the purposes of reporting and costing.

A particular way of describing sets of Objects for the purpose of reporting or costing. For example, Brand and Package Type are both Characteristics.

The allowable set of values for a particular Characteristic. That is, the Characteristic domain. The brand Characteristic for example, contains all of the possible brand values.

Figure 8.4: Subset of SAP/R3 Classifications

Business Rules (listed in the order we would typically walk someone through the model):

- Each Class may contain one or many Classes.
- Each Class may belong to one Class.
- Each Class must contain many Characteristics.
- Each Characteristic must belong to many Classes.
- Each Characteristic may contain one or many Characteristic Values.
- Each Characteristic Value must belong to one Characteristic.
- Each Class must organize many Objects.
- Each Object must be organized by many Classes.
- Each Object must be assigned many Characteristic Values.
- Each Characteristic Value must be assigned to many Objects.

This SAP/R3 ASAM was used for educational purposes, as well as to gauge the effort to customize the application. Notice that on this particular model, showing the definitions directly on the model helps the reader to understand the concepts, as terms such as Object and Class can be ambiguous. This model was first used in a joint IT and business department meeting where the goal was to clearly explain SAP/R3 Classifications in 30 minutes or less. This was not an easy task as SAP/R3 is a very complicated system, and showing the audience the underlying 350 database tables within the Classifications area would most likely have led to the audience running screaming from the room. Therefore, this model was produced to get the main concepts across to the audience. Somewhere hidden in those 350 database tables are these four key concepts and accompanying business rules. The model was extremely well-received, and I know from the comments and questions from the audience that both the business and IT folks got it.

We can also build a dimensional ASAM, such as the example in Figure 8.5 which builds on our ice cream example.

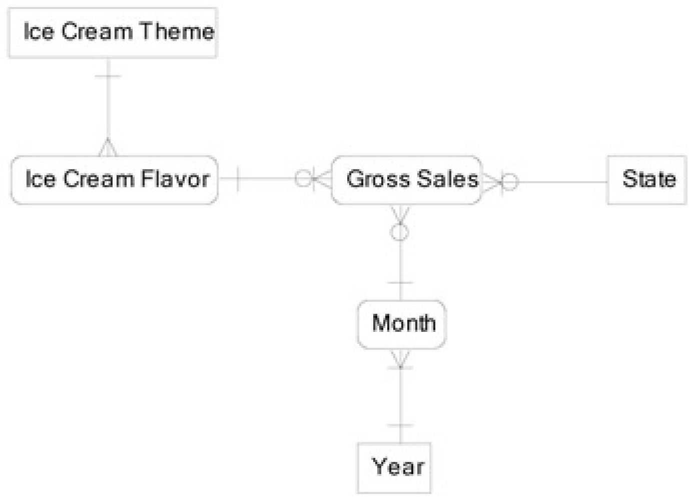


Figure 8.5: Dimensional ASAM example

Subject area definitions:

Gross Sales	The total dollar amount charged for all goods or products sold or distributed. Also included in gross sales is sales tax. Returns from melted ice cream are not deducted.
Ice Cream Flavor	The distinctive taste of one ice cream product over another. Flavors are recognized and selected by consumers. Examples include Chocolate, Vanilla, and Strawberry.
Ice Cream Theme	A group of ice cream flavors that share at least one common property. For example, the Brownie Ice Cream and Cake Batter Ice Cream flavors are grouped into the Cake Ice Cream Theme.
Month	One of the twelve major subdivisions of the year.
State	A subset of the United States; one of the fifty states.
Year	A period of time containing 365 days (or 366 days in a leap year).

Navigation paths (listed in the order a user would typically navigate through the model):

"I need to see Gross Sales for each Theme and Year. If there are any surprises, I will need to drill down into the details. That is, drill down from Theme to Flavor, from Year to Month, and add State to my queries."

Notice that with a dimensional model, the focus is more on navigation than on business rules, as in a relational model.

Comparison Subject Area Model (CSAM)

The CSAM is a subject area model that shows how something new fits within an existing environment. It is used primarily for impact analysis and issue resolution. CSAMs build on two or more ASAMs, in which one or more ASAMs represent an existing application environment and (optionally) one or more ASAMs represent proposed changes to an environment.

The CSAM identifies gaps, touch points, and overlaps. It compares something new with something in place to see whether there are gaps or redundancies. Here are some examples where a CSAM would be useful:

- **Proposed data mart within current data warehouse architecture.** The CSAM will show how the new data mart requirements fit into what currently exists within the data warehouse.
- **Proposed new functionality to an existing operational or data mart application.** The CSAM will highlight the new functionality and show how it fits in with current functionality.
- **Replacing an application.** The CSAM can show the overlap at a subject area level when you are replacing an existing application with a new application.

The CSAM also helps estimate the development effort. Because we are showing the overlap and gaps that a new application will cause within our environment, we can more easily develop a high-level estimate for the development effort. For example, superimposing a new reporting application over an existing data warehouse environment can help us find areas of the reporting application where the programming effort will be greater than in those areas already existing in the warehouse.

The CSAM requires the most effort to build, but of the three types of subject area models, it can provide the most benefits. It requires the most effort because there is more than one viewpoint to represent on a model. More than one perspective often leads to more than one interpretation of a subject area or the rules between subject areas. It takes time to identify these differences and resolve, or at least document them. The CSAM provides the most benefits because we are showing impact or issues at a high-level of granularity. What's more, it is likely that this high-level, cross-application view has not been captured and explained with such clarity previously. Figure 8.6 includes an example of a CSAM.

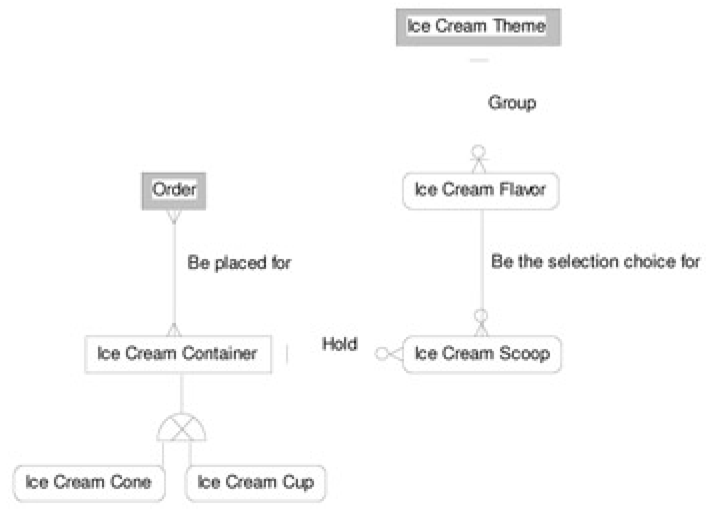


Figure 8.6: CSAM example

In the CSAM in Figure 8.6, we are capturing how a proposed new data mart would impact an existing data warehouse. Building on our ice cream example, the current data warehouse for this ice cream store currently contains those subject areas that are not shaded in gray. That is, the data warehouse currently contains Ice Cream Cone, Ice Cream Container, Ice Cream Cup, Ice Cream Flavor, and Ice Cream Scoop. The dimensional ASAM example from Figure 8.5 is a proposed data mart. All of the concepts on this proposed data mart are in the ice cream store data warehouse with the exception of those two terms shaded in gray: Ice Cream Theme and Order. (Note that the concepts of Month, Year, State, and Gross Sales from Figure 8.5 are derived from detailed orders. Therefore, Order in Figure 8.6 includes these four concepts.) This CSAM is an excellent tool to explain to the data warehouse and data mart stakeholders what the impact (and therefore the cost and benefit) of creating this new data mart within the existing data warehouse would be.

Subject area definitions:

Ice Cream Cone	An edible container which can hold one or more ice cream scoops.
Ice Cream Container	A holder of ice cream that is either edible, in the case of an ice cream cone, or recyclable, in the case of an ice cream cup.
Ice Cream Cup	A recyclable container which can hold one or more ice cream scoops.
Ice Cream Flavor	The distinctive taste of one ice cream product over another. Flavors are recognized and selected by consumers. Examples include Chocolate, Vanilla, and Strawberry.
Ice Cream Scoop	A single ball-shaped unit of ice cream formed in an ice cream scooper and packaged in one ice cream container.
Ice Cream Theme	A collection of ice cream flavors that have been grouped together because they share at least one common property. For example, the Brownie Ice Cream and Cake Batter Ice Cream flavors are grouped into the Cake Ice Cream Theme.

Order	Also known as an ice cream sale, an order is when a consumer purchases one or more of our products in a single visit to our ice cream store.

Business Rules (listed in the order we would typically walk someone through the model):

- Each Ice Cream Container may hold one or many Ice Cream Scoops.
- Each Ice Cream Scoop must be held in one Ice Cream Container.
- Each Ice Cream Container may be either an Ice Cream Cone or an Ice Cream Cup.
- Each Ice Cream Cone is an Ice Cream Container.
- Each Ice Cream Cup is an Ice Cream Container.
- Each Ice Cream Flavor may be the selection choice for one or many Ice Cream Scoops.
- Each Ice Cream Scoop must contain one Ice Cream Flavor.
- Each Ice Cream Theme may group one or many Ice Cream Flavors.
- Each Ice Cream Flavor must be grouped by one Ice Cream Theme.
- Each Order must be placed for many Ice Cream Containers.
- Each Ice Cream Container must appear on many Orders.

How to Build a Subject Area Model

Following is a brief summary of the ten steps to complete a SAM (for more on this approach, please refer to *Data Modeling for the Business: A Handbook for Aligning the Business with IT using High-Level Data Models*, ISBN 9780977140077):

1. **Identify model purpose.** Before starting any data modeling effort, first identify why the model needs to be built. The underlying reason for building a data model is communication. We build data models so we can ensure that we and others have a precise understanding of terminology and business rules. In this step, we identify our communication needs. What are we communicating? Who are we communicating it to? Always "begin with the end in mind", as Stephen Covey would say. The most popular reason for building a SAM is to gain understanding into an existing area of the business. We can model a department such as Sales or Accounting, a line of business such as Home Mortgages, or even the entire organization. The SAM is also a very good place to start capturing the concepts and business rules for a new application. This way, terminology, rules, and definitions can be agreed upon prior to detailed project analysis.
2. **Identify model stakeholders.** Document the names and departments of those who will be involved in building the SAM, as well as those who will use it after its completion. Roles include architects, analysts, modelers, business users, and managers. Both business and IT roles are required for success. I remember one project I worked on where only IT people showed up to build the SAM. As enthusiastic as we all were to build the model and get support for the project, it was a losing effort because without the business, the model lost credibility and the project, therefore, lost financial support.
3. **Inventory available resources.** Leverage the results from Step 2, as well as identify any documentation that could provide useful content to the SAM. The two types of resources are people and documentation. People include both business and IT resources. Business people may be management and/or knowledge users. IT resources can span the entire IT spectrum, from analysts through developers, from program sponsors to team leads. From the people

side, this step can be as simple as assigning actual people to the builder roles in Step 2. For example, if we are building a SAM of the manufacturing section of our business, Bob, the business analyst who spent 20 years of his career working in the plant, would be an excellent resource to assign as a builder. Documentation includes systems documentation and requirements documents. Systems documentation can take the form of documentation for a packaged piece of software, or documentation written to support a legacy application. Requirements documents span business, functional, and technical requirements and can be an essential input to building the SAM.

4. **Determine type of model.** Decide whether the BSAM, ASAM, or CSAM would be most appropriate. Base your selection on the purpose of the model and the available resources.
5. **Select approach.** There are three approaches for building a subject area model: top-down, bottom-up, and hybrid. Even though these approaches sound completely different from each other, they really have quite a lot in common. In fact, the major difference between the approaches lies in the initial information-gathering step. The top-down approach begins with a purely business needs perspective. We learn how the business works and what the business needs from the business people themselves, either through direct meetings with them, or indirectly through requirements documents and reports. The business is allowed to dream - they should aim for the sky. All requirements are considered possible and doable. The bottom-up approach, on the other hand, temporarily ignores what the business needs and instead focuses on the existing systems environment. We build an initial subject area model by studying the systems that the business is using today. It may include operational systems that run the day-to-day business or reporting systems that allow the business to view how well the organization is doing. Once the existing systems are understood at a high level, new concepts may be introduced or existing concepts can be modified to meet the business needs. The hybrid approach completes the initial information gathering step by usually starting with some top-down analysis and then some bottom-up analysis, and then some top-down analysis, etc., until the information gathering is complete. First, there is some initial top-down work to understand project scope and high level business needs. Then we work bottom-up to understand the existing systems. The whole process is a constant loop of reconciling what the business needs with what information is available.
6. **Complete audience-view SAM.** Produce a SAM using the terminology and rules that are clear to those who will be using the model. This is the first high-level model we build. Our purpose is to capture the viewpoint of the model's audience without complicating information capture by including how their perspective fits with other departments or with the organization as a whole. Our next step will reconcile the deliverable from this step with enterprise terminology. Here our purpose is just to capture *their* view of the world. The initial model can be created by standing at a white board or flipchart with the users. Sometimes 30 minutes at a white board with a business user can reveal more about what they want than spending hours reading requirements documents.
7. **Incorporate enterprise terminology.** Now that the model is well-understood by the audience, ensure the terminology and rules are consistent with the organization view. To build the enterprise perspective, modify the audience model to be consistent with enterprise terminology and rules. Ideally, this enterprise perspective is captured within an enterprise data model. You might have to schedule meetings between the people whose view is captured in the audience-specific model and those who share the enterprise perspective to resolve any terminology or definition issues.

8. **Signoff.** Obtain approval from the stakeholders that the model is correct and complete. After the initial information gathering, make sure the model is reviewed for data modeling best practices, as well as for meeting the requirements. The signoff process on a SAM does not require the same formality as signoff on the physical design, but it should still be taken seriously. Usually, email verification that the model looks good will suffice. Validating whether the model has met data modeling best practices is often done by applying the Data Model Scorecard®, which will be discussed in Chapter 12.
9. **Market.** Similar to introducing a new product, advertise the data model so that all those who can benefit from it know about it. If the model has a relatively small scope, it is possible that the users who helped you build and validate the model are also the only ones who need to know about it, so marketing requires little effort. However, most SAMs span broad areas, so it is essential to let the appropriate business and IT resources know it is available to them. Techniques include 'Lunch-and-Learns', advertisements on corporate intranets, and presenting the model at periodic management meetings.
10. **Maintain.** SAMs require little maintenance, but they are not maintenance-free. Make sure the model is kept up-to-date. The SAM will not change often, but it will change and we need to have formal processes for updating this model to keep it up to date. What works well is to have two additional steps in your software methodology: 1. Borrowing from a SAM, and 2. Contributing back to the SAM. After the SAM is complete, and before starting a logical data model, for example, the SAM should be used as a starting point for further modeling. The second step in the methodology requires taking all of the learnings from the project and making sure all new concepts are incorporated back into the SAM.

Exercise 8: Building a SAM

Identify an area within your organization that is in desperate need of a SAM and build it for them using the ten-step approach from this chapter.

Key Points

- A subject area is a concept that is both basic and critical to your audience.
- A subject area model is a set of symbols and text that represents key concepts and the rules binding these key concepts for a specific business or application scope and for a particular audience.
- The BSAM is a subject area model of a defined portion of the business.
- The ASAM is a subject area model of a defined portion of a particular application, and may be relational or dimensional.
- The CSAM is a subject area model that shows how something new fits within an existing framework.
- Follow the ten-step approach to building a SAM.

Chapter 9: What are Logical Data Models?

Overview

What does business need?
Forget the technology
Enter logical

The highlighted row in Table 9.1 shows the focus of this chapter, which is the logical data model.

Table 9.1: The LDM is the focus of this chapter

➡Open table as spreadsheet

	Relational	Dimensional
SAM	'One-pager' on how something works	'One-pager' on what is monitored
LDM	**Detailed business solution on how something works**	**Detailed business solution on what is monitored**
PDM	Detailed technical solution on how something works	Detailed technical solution on what is monitored

A logical data model (LDM) takes the business need defined on a subject area model down to the next level of a business solution. That is, once you understand at a broad level the scope of an effort and what business people require to solve their problem, the next step is to come up with a solution for them in the form of a LDM. The logical data model is explained, along with a comparison of relational and dimensional mindsets. Then, for relational models, the techniques of normalization and abstraction are discussed. I conclude by answering Frequently Asked Questions (FAQ) on dimensional modeling, which leads to explaining terms such as conformed dimensions and factless facts.

Logical Data Model Explained

A logical data model (LDM) is a business solution to a business problem. It is how the modeler captures the business requirements without complicating the model with implementation concerns such as software and hardware. Recall the camera analogy discussed earlier. The negative returned with a developed set of photographs represents the logical data model because it is a perfect view of the subject matter.

On the subject area model, we might learn, for example, what the terms, business rules, and scope would be for a new order entry system. After understanding the requirements for the order entry system, we create a LDM containing all of the data elements and business rules needed to deliver the business solution system. For example, the subject area model will show that a Customer places many Orders. The LDM will capture all of the details behind Customer and Order, such as the customer's name, their address, the order number and what is being ordered.

While building the LDM, questions or issues may arise having to do with specific hardware or software such as:

- How can we retrieve this information in less than 5 seconds?
- How can we make this information secure?
- There is a lot of information here. What is the best way to manage storage space?

These questions focus on hardware and software. Although they need to be documented, they are not addressed until we are ready to start the physical data model. The reason these questions depend on technology is because if hardware and software were infinitely efficient and secure, these questions would never be raised.

Comparison of Relational with Dimensional Logical Models

There are both relational and dimensional logical data models. Relational modeling is the process of capturing how the business works, while dimensional modeling is the process of capturing the information the business needs to monitor how well it is doing. Relational modeling captures how the business works and contains business rules, such as "A Customer must have at least one Account", or a "Product must have a **Product Short Name**." Dimensional modeling focuses on capturing and aggregating the metrics from daily operations that enable the business to evaluate how well it is doing by manipulating the numbers. For example, examining the measure **Gross Sales Amount** at a day level and then, after getting the answer, looking at **Gross Sales Amount** at a month or year level, or at a product or brand level, or a city or country level. The dimensional model is all about playing with numbers.

Recall the Ice Cream CSAM from Figure 8.6, a subset of which appears in Figure 9.1.

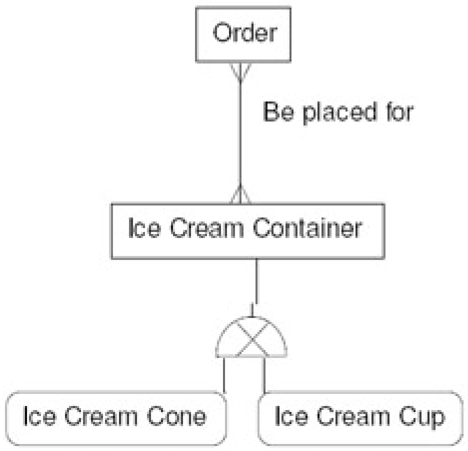


Figure 9.1: Ice cream CSAM subset

Figure 9.2 contains the relational logical data model for this CSAM.

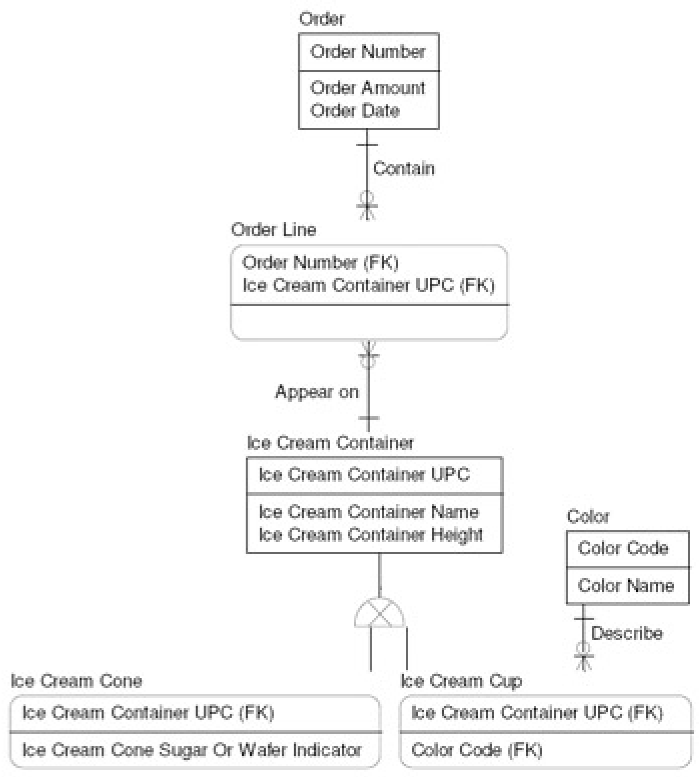


Figure 9.2: Ice cream relational logical data model

Data element definitions:

Color Code	The short name of one of the primary colors, such as 'b' for 'blue'.
Color Name	The full name of one of the primary colors, such as 'blue', 'red', or 'yellow'.
Ice Cream Cone Sugar Or Wafer Indicator	Determines whether the ice cream cone is made of a sugar-based or wafer-based material. Contains either the value 'S' for 'Sugar' or 'W' for 'Wafer'.
Ice Cream Container Height	The height, in inches, of the container. Used by inventory to determine how many containers can fit on a storage shelf.
Ice Cream Container Name	The name our store employees use for communicating with consumers on what they would like to order. Examples range from the smallest, called "Kid's Cup", to the largest size, called "Kitchen Sink".
Ice Cream Container UPC	UPC (Universal Product Code) is a numeric code used to identify a specific product across the ice cream industry.
Order Amount	The total dollar amount of the order, including sales tax.
Order Date	The date the order was placed.
Order Number	The unique number assigned to each order placed by a consumer. It is a numeric sequence number that never repeats.

Business Rules (listed in the order we would typically walk someone through the model):

- Each Order may contain one or many Order Lines.
- Each Order Line must belong to one Order.
- Each Ice Cream Container may appear on one or many Order Lines.
- Each Order Line must reference one Ice Cream Container.
- Each Ice Cream Container may be either an Ice Cream Cone or an Ice Cream Cup.
- Each Ice Cream Cone is an Ice Cream Container.
- Each Ice Cream Cup is an Ice Cream Container.
- Each Color may describe one or many Ice Cream Cups.
- Each Ice Cream Cup must be described by one Color.

A relational logical data model captures the business solution for how part of the business works. On this ice cream relational logical data model, we are representing the properties and rules for part of an ice cream ordering system, including Order, Order Line, and Ice Cream Container. Our solution for example, allows us to capture whether an ice cream cone is a sugar or wafer cone, but does not allow us to capture the color of the ice cream cone. We can only capture the color for cups, not cones.

Two additional notes on the model in Figure 9.2:

- The primary key for a subtype must have the same primary key as its supertype. Given "Each Ice Cream Cone is an Ice Cream Container" and "Each Ice Cream Cup is an Ice Cream Container", it makes sense for an Ice Cream Cone to have the same primary key as an Ice Cream Container, and an Ice Cream Cup to have the same primary key as an Ice Cream Container.
- There are more details on the logical data model. Whereas the subject area model contained a many-to-many relationship between Order and Ice Cream Container, the logical model contains more details, including the new entity Order Line, which resolves the many-to-many relationship between Order and Ice Cream Container. Also, the Color logical entity was not considered basic and critical and therefore did not need to be shown on the subject area model. In fact, it is not uncommon to have ten or more logical entities capture the details behind one subject area.

A dimensional logical data model captures all of the information needed to answer business questions where the answers to the questions are measures, and these measures need to be reported at different levels of granularity. Granularity means the level of detail of the measures, such as by a particular level of calendar including day, month, or year, or by a particular level of geography, including city, region, or country. Here are some examples of business questions that focus on measures:

- What is my **Gross Sales Amount** by Region, Product, and Month? Now what about by Region and Year? How about by Product Brand and Quarter?
- How many Students majored in Computer Science this Semester? How about for the whole Year?

- What is our **Member Growth Rate** by Region over the over the last five Years? How about by Country over the last ten Years?

Figure 9.3 contains the dimensional logical data model that is based on the ice cream example from Figure 8.5.

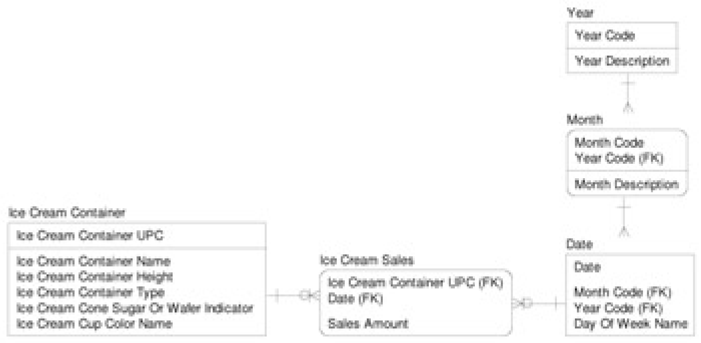


Figure 9.3: Ice cream dimensional logical data model

Sales Amount is captured at the Date and Ice Cream Container level and then the user (or reporting tool) can navigate and look at **Sales Amount** at different levels of detail. "What was our Sales of Sugar Ice Cream Cones in August 2009? Oh, that number is so low. What was our Sales of Sugar Ice Cream Cones each date in August 2009? Or how about Sales for all of 2009 for all Ice Cream Containers?" These are examples of some of the questions that can be answered. Note that all of these questions focus on measures, in this case, **Sales Amount**.

Although the symbols are very similar across both the relational model in Figure 9.2 and the dimensional model in Figure 9.3, the modeling mindset required is very different. With the relational model we ask "How does this business work?" With the dimensional model we ask "What does the business want to monitor?" On a relational model, we have the building blocks of entity, data element, and relationship. On a dimensional model, we have similar symbols, but they represent different concepts:

- **Measure.** A data element that may be manipulated in or is the result of a calculation (e.g., sum, count, average, minimum, maximum). **Sales Amount** is an example of a measure in our ice cream example.
- **Meter.** A meter is an entity containing a related set of measures. It is not a person, place, event, or thing, as we find on the relational model. Instead, it is a bucket of common measures. As a group, common measures address a business concern, such as Profitability, Employee Satisfaction, or Sales. The meter is so important to the dimensional model that the name of the meter is often the name of the application.
- **Grain.** The grain is the meter's lowest level of detail. It should be low enough that the answers to all of the business questions within the scope of the dimensional model are derivable. It is generally a good practice to define the measures and grain as early as possible in the requirements process. The grain in our ice cream example is Ice Cream Container and Date.
- **Dimension.** A dimension is reference information whose purpose is to add meaning to the measures. All the different ways of filtering, sorting, and summing measures make

use of dimensions. Dimensions are often, but not exclusively, hierarchies. A hierarchy is when a higher level breaks down into many lower levels, but a lower level rolls up into only one higher level. Our calendar structure in Figure 9.3 is an example of a hierarchy, because every Date rolls up into a single Month and every Month rolls up into a single Year. Also, every Year contains many Months and every Month contains many Dates. Ice Cream Container is an example of a dimension that is not a hierarchy.

- **Dimensional level.** A dimensional level is one level within the hierarchy of a dimension, such as Month within the Calendar dimension. Dimensional levels are used to facilitate calculating measures at the level(s) the business needs to see. They are not built based on how the business works.
- **Dimensional attribute.** The properties within a dimension, such as Ice Cream Container Height in the Ice Cream Container dimension.

We've emphasized several times that relational modeling captures how something works and dimensional captures what is being monitored. Let's look at this distinction in another way - there are three main differences between relational and dimensional models: focus, lines, and scope:

- **Focus.** A dimensional model is a data model whose only purpose is to allow efficient and user-friendly filtering, sorting, and summing of measures. A relational model, on the other hand, focuses on supporting a business process. Dimensional models are appropriate when there is a need to massage numbers, such as by summing or averaging. The reason the dimensional model should be limited to numbers is because its design allows for easy navigation up and down hierarchy levels. When traversing hierarchy levels, measures may need to be recalculated for the hierarchy level. For example, a **Gross Sales Amount** of $5 on a particular date might be $100 for the month in which that date belongs.
- **Lines.** The relationship lines on a dimensional model represent navigation paths instead of business rules, as in a relational model. Let creativity drive your dimensional structures. The relationships in a dimensional hierarchy do not have to mimic their relational counterparts. You should build dimensions to meet the way the business users think.
- **Scope.** The scope of a dimensional model is a collection of related measures that together address a business concern, whereas in a relational model, the scope may be a broad business process, such as order processing or account management. For example, the metrics **Number of Product Complaints** and **Number of Product Inquiries** can be used to gauge product satisfaction.

Normalization Explained

When I turned 12, I received a trunk full of baseball cards as a birthday present from my parents. I was delighted, not just because there may have been a Hank Aaron or Pete Rose buried somewhere in that trunk, but because I loved to organize the cards. I categorized each card according to year and team. Organizing the cards in this way gave me a deep understanding of the players and their teams. To this day, I can answer many baseball card trivia questions.

Normalization, in general, is the process of applying a set of rules with the goal of organizing *something*. I was normalizing the baseball cards according to year and team. We can also apply a set of rules and normalize the data elements within our organizations. The rules are based on how the business works, which is why normalization is the primary technique used in building the relational logical data model.

Just as those baseball cards lay unsorted in that trunk, our companies have huge numbers of data elements spread throughout departments and applications. The rules applied to normalizing the baseball cards entailed first sorting by year, and then by team within a year. The rules for normalizing our data elements can be boiled down to a single sentence:

Make sure every data element is single-valued and provides a fact completely and only about its primary key. The underlined terms require more of an explanation.

'Single-valued' means a data element must contain only one piece of information. If **Consumer Name** contains **Consumer First Name** and **Consumer Last Name,** for example, we must split **Consumer Name** into two data elements - **Consumer First Name** and **Consumer Last Name**.

'Provides a fact' means that a given primary key value will always return no more than one of every data element that is identified by this key. If a **Customer Identifier** value of '123' for example, returns three customer last names ('Smith', 'Jones', and 'Roberts'), this violates this part of the normalization definition.

'Completely' means that the minimal set of data elements that uniquely identify an instance of the entity is present in the primary key. If, for example, there are two data elements in an entity's primary key, but only one is needed for uniqueness, the data element that is not needed for uniqueness should be removed from the primary key.

'Only' means that each data element must provide a fact about the primary key and nothing else. That is, there can be no hidden dependencies. For example, assume an Order is identified by an **Order Number**. Within Order, there are many data elements, including **Order Scheduled Delivery Date, Order Actual Delivery Date**, and **Order On Time Indicator. Order On Time Indicator** contains either a 'Yes' or a 'No', providing a fact about whether the **Order Actual Delivery Date** is less than or equal to the **Order Scheduled Delivery Date. Order On Time Indicator,** therefore, provides a fact about **Order Actual Delivery Date** and **Order Scheduled Delivery Date**, not directly about **Order Number**, so it should be removed from the normalized model. **Order On Time Indicator** is an example of a derived data element, meaning it is calculated. Derived data elements are removed from a normalized model.

So, a general definition for normalization is that it is a series of rules for organizing something. The series of rules can be summarized as: *Every data element is single-valued and provides a fact completely and only about its primary key*. An informal definition I frequently use for normalizing is: *A formal process of asking business questions*. We cannot determine if every data element is single-valued and provides a fact completely and only about its primary key unless we understand the data. To understand the data we usually need to ask lots of questions. Even for an apparently simple data element such as **Phone Number,** for example, we can ask many questions:

- Whose phone number is this?
- Do you always have to have a phone number?
- Can you have more than one phone number?
- Do you ever recognize the area code as separate from the rest of the phone number?
- Do you ever need to see phone numbers outside a given country?
- What type of phone number is this? That is, is it a fax number, mobile number, etc.?

To ensure that every data element is single-valued and provides a fact completely and only about its primary key, we apply a series of rules or small steps, where each step (or level of normalization) checks something that moves us towards our goal. Most data professionals would agree that the full set of normalization levels is the following:

- first normal form (1NF)
- second normal form (2NF)
- third normal form (3NF)
- Boyce/Codd normal form (BCNF)
- fourth normal form (4NF)
- fifth normal form (5NF)

Each level of normalization includes the lower levels of rules that precede it. If a model is in 5NF, it is also in 4NF, BCNF, and so on. Even though there are higher levels of normalization than 3NF, many interpret the term "normalized" to mean 3NF. This is because the higher levels of normalization (that is, BCNF, 4NF, and 5NF) cover specific situations that occur much less frequently than the first three levels. Therefore, to keep things simple, this chapter focuses only on first through third normal forms.

Normalization provides a number of important benefits: (NOTE: In this section, the term data model may also represent the physical database that will be implemented from the data model)

- **Stronger understanding of the business.** The process of normalization ensures that we ask many questions about the data elements so that we know we are assigning them to entities correctly. The answers to our questions give us insight into how things work. Recently, for example, I normalized a Claims and Provider file for a medium-sized insurance company. I kept a record of my questions and their responses to them. After normalization was complete, I realized I had asked over 100 questions! I learned quite a bit about claims and providers from this process.
- **Greater application stability.** Normalization leads to a model that mimics the way in which the business works. As the business goes about its daily operations, the application receives data according to the rules that govern the business. The normalized model developed for the application leads to a database that has been structured to match these rules. Therefore, the system runs smoothly as long as the rules in the business match the rules documented throughout the normalization process. If, for example, more than one

person can own one or more accounts, the model—and therefore the application—will accommodate this fact.

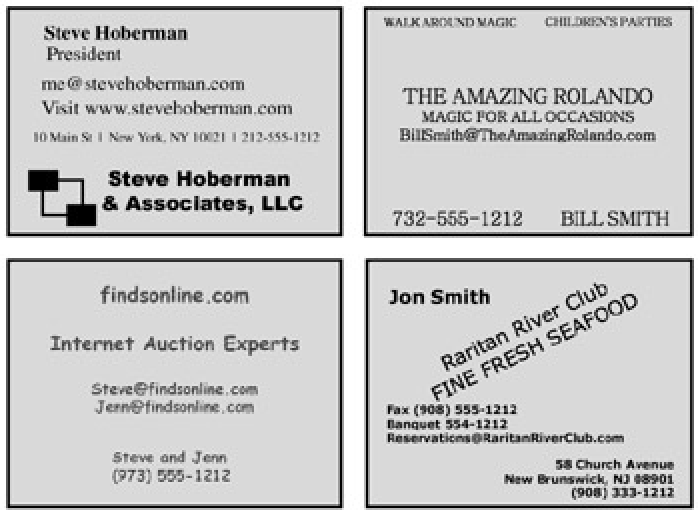
- **Less data redundancy.** Each level of normalization removes a type of *data redundancy* from the model. Data redundancy occurs when the same information appears more than once in the same model. As redundancy is removed, changing the data becomes a quicker process, because there is less of it to update or insert. If **Person Last Name** appears once on a model and Mary's last name changes, the application only has to update her last name in one place.
- **Better data quality.** By reducing redundancy and enforcing business rules through relationships, the data are less likely to get out of synch or violate these business rules. If an account must have at least one account owner, the data model can prevent accounts with invalid or missing account owners from occurring.
- **Faster building of new models.** A degree of common sense is applied to the place to which data elements are assigned during the normalization process. Therefore, it becomes easier to identify and reuse normalized structures on a new model. All data elements that require **Account Identifier** for uniqueness appear in Account. The result is more consistency across models and less time developing applications.

Initial Chaos

We have all put together puzzles at one time or another. After you open the box and dump out the pieces, the sight can be overwhelming: hundreds, maybe thousands, of tiny pieces in a large pile. The pile is in a state of chaos. As we pick each piece up and examine it, we understand its properties in the form of its shape. We lack knowledge at this point about how these puzzle pieces connect with each other. We begin the process of fitting the pieces together and, after much effort, we complete our masterpiece. Each puzzle piece is in its proper place.

The term *chaos* can be applied to any unorganized pile, including data elements. We may have a strong understanding of each of the data elements, such as their name and definition, but we lack knowledge about how the data elements fit together. We understand each piece of the puzzle, but we do not yet understand the connections between the pieces. In our business card example, we do not know the relationship between **Mailing Address** and **Phone Number** before normalizing. Just as we need to determine the appropriate place for each piece within our puzzle, we need to determine the appropriate place for each data element within our model.

Recall our four business cards from Figure 1.2, repeated here as Figure 9.4.


Steve Hoberman
President
me@stevehoberman.com
Visit www.stevehoberman.com
10 Main St | New York, NY 10021 | 212-555-1212
Steve Hoberman
& Associates, LLC

WALK AROUND MAGIC CHILDREN'S PARTIES
THE AMAZING ROLANDO
MAGIC FOR ALL OCCASIONS
BillSmith@TheAmazingRolando.com
732-555-1212 BILL SMITH

findsonline.com
Internet Auction Experts
Steve@findsonline.com
Jenn@findsonline.com
Steve and Jenn
(973) 555-1212

Jon Smith
Raritan River Club
FINE FRESH SEAFOOD
Fax (908) 555-1212
Banquet 554-1212
Reservations@RaritanRiverClub.com
58 Church Avenue
New Brunswick, NJ 08901
(908) 333-1212

Figure 9.4: Four business cards from my nightstand

In this business card example, chaos starts off as one pile of data elements. In other words, all the data elements can be initially assigned to one entity, the business card itself. See Figure 9.5.

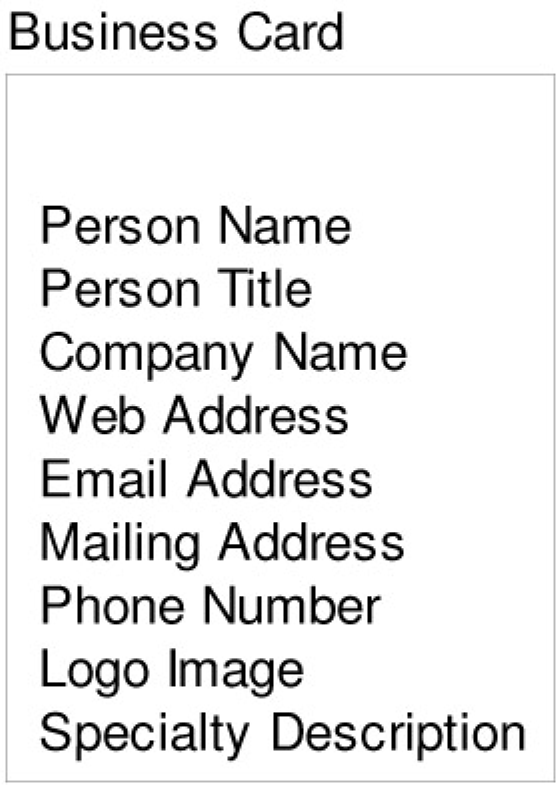


Figure 9.5: Initial set of data elements in chaotic state

As was noted earlier, it is sometimes easier to think of an entity as a set of spreadsheet columns. Each data element in the entity is a separate column, and the rows in the spreadsheet represent the entity instances. Table 9.2 is a spreadsheet showing the data elements from Figure 9.5 as columns and the four business cards from Figure 9.4 as rows.

Table 9.2: Four sample business cards in spreadsheet format

➡Open table as spreadsheet

Card	Person Name	Person Title	Company Name	Web Address	Email Address	Mailing Address	Phone Number	Logo Image	Specialty Description
1	Steve Hoberman	President	Steve Hoberman & Associates, LLC	www.stevehoberman.com	me@stevehoberman.com	10 Main St New York, NY 10021	212-555-1212	Entity Model.jpg	
2	Steve Jenn		findsonline.com	findsonline.com	Steve@findsonline.com Jenn@findsonline.com		(973) 555-1212		Internet auction experts
3	Bill Smith		The Amazing Rolando		BillSmith@TheAmazingRolando.com		732-555-1212		Magic for all occasions Walk around magic Children's parties
4	Jon Smith		Raritan River Club		Reservations@RaritanRiverClub.com	58 Church Avenue New Brunswick, NJ 08901	(908) 333-1212 (908) 555-1212 554-1212		Fine fresh seafood

We hit a very interesting data modeling challenge in this step. It is a challenge that many of us face on the job when we first start a modeling project. The challenge is whether to model the medium or the content of the medium. In other words, do we model the form, report, or in this case business card? Or do we model the contents of the form, report, or business card?

In one of my data modeling classes, the students need to model a survey form. Some model the form itself, with a prominent entity in the center called Survey and supporting entities surrounding it. Others model the content of the survey, modeling the questions and creating data elements for the responses.

We need to ask a basic question to know whether we need to model the medium or the contents of the medium: *Why are we building this data model?* If we are building it for a business card company to automate the process of printing business cards, then the business card entity is appropriate. If we are building it for an organization or person to manage their contact information, then the business card is no longer the important concept. Rather, the contents of the business card are important and the business card serves as a grouping mechanism or source for our information.

To make the business card exercise even more fun, let's assume we are modeling this for a contact management organization, and the model you build will be the foundation for a new contact management application. In other words, we will model the content instead of the medium. Therefore, let's update the model as shown in Figure 9.6.

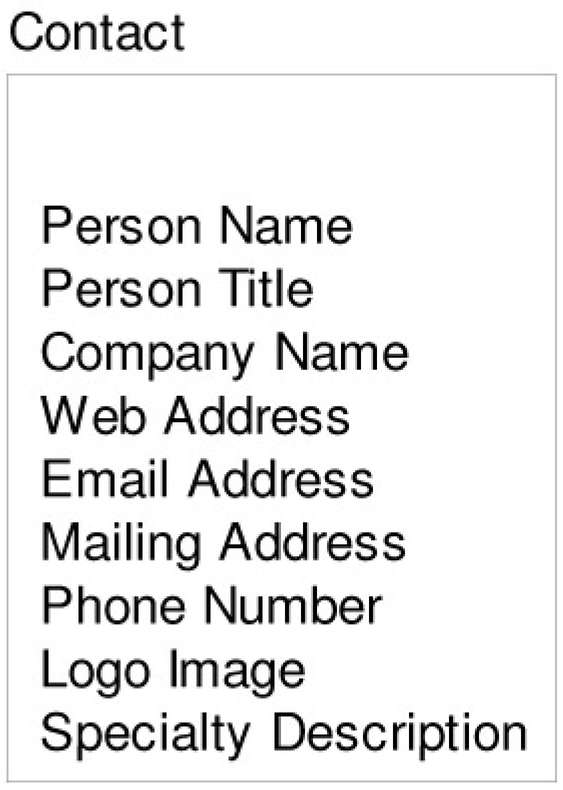


Figure 9.6: Contact management data elements in chaotic state

Normalizing is a process of asking questions. After asking the key question on model purpose, we now have another important question: What is a Contact? This is where we realize how important definitions are to the analysis and modeling process. Contact can mean different things to different people or departments, much the same way as Customer can mean different things to different people or departments. Is contact the person in an organization whose phone number and email address we would like to capture? Is it the organization's phone number and email address itself? Or is contact the person's phone number or email address, and the person or organization becomes the contactee?

After hours of talking to people in different departments in the organization building this contact management application, we have come up with this definition of Contact:

- *A person, a company, or a person at a company who our organization cares about knowing how to reach.*

Now that we know what a Contact is, how would a businessperson request a specific Contact? We need to ask how a Contact is identified. We need something that, at least initially, makes a contact unique. If we are replacing an existing contact management system, we can investigate what makes a contact unique in that system and use the same primary key here. If this is a new contact management system, we need to interview businesspeople and consult requirements documents to determine what data element or data elements can uniquely identify a contact.

Let's assume that this is for a new system and the manager of the department building the system would like the system to create a new unique identifier for each contact. The manager would like this **Contact Id** combined with the contact's email address to guarantee uniqueness. See Figure 9.7 for the Contact model with this new primary key added.

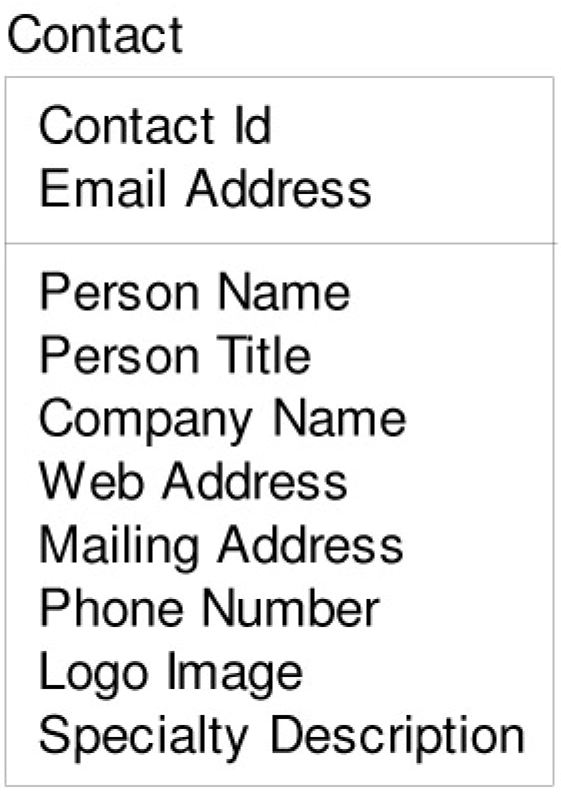


Figure 9.7: Contact entity with primary key

In normalizing, you have to ask a business expert many questions or examine a lot of actual data to answer the questions themselves. Table 9.3 shows the same data from Table 9.2 with the rows divided to show the impact of defining a contact by **Contact Id** and **Email Address**. Note that a business card can have more than one contact. Note also that **Contact Id** is just a meaningless number and therefore the values do not have to be a sequential counter.

Table 9.3: Four sample business cards in spreadsheet format segmented by Contact

Open table as spreadsheet

Card	Contact Id	Email Address	Person Name	Person Title	Company Name	Web Address	Mailing Address	Phone Number	Logo Image	Specialty Description
1	123	me@stevehoberman.com	Steve Hoberman	President	Steve Hoberman & Associates, LLC	www.stevehoberman.com	10 Main St New York, NY 10021	212-555-1212	EntityModel.jpg	
2	54	Steve@findsonline.c	Steve		findsonli	findsonline.co		(973)		Intern

Table 9.3: Four sample business cards in spreadsheet format segmented by Contact

Open table as spreadsheet

Card	Contact Id	Email Address	Person Name	Person Title	Company Name	Web Address	Mailing Address	Phone Number	Logo Image	Specialty Description
		om			ne.com	m		555-1212		et auction experts
	58	Jenn@findsonline.com	Jenn		findsonline.com	findsonline.com		(973) 555-1212		Internet auction experts
3	42	BillSmith@TheAmazingRolando.com	Bill Smith		The Amazing Rolando			732-555-1212		Magic for all occasions Walk around magic Children's parties
4	14	Reservations@RaritanRiverClub.com	Jon Smith		Raritan River Club		58 Church Avenue New Brunswick, NJ 08901	(908) 333-1212 (908) 555-1212 554-1212		Fine fresh seafood

Now we are going to remove redundancies and assign each data element to its proper entity, following the first three levels of normalization. Recall our initial definition of normalization: *Every data element is single-valued and must provide a fact completely and only about its primary key*. To formalize this slightly, and to more accurately fit the description for the first three levels of normalization, we can rephrase this definition of normalization into: *Every data element must provide a fact about the key, the whole key, and nothing but the key*. First Normal Form (1NF) is the "Every data element must provide a fact about the key" part. Second Normal Form (2NF) is "the whole key" part, and Third Normal Form (3NF) is the "nothing but the key" part.

First Normal Form (1NF)

1NF ensures that "Every data element must provide a fact about the key". 'Key' refers to the primary key of the entity. 'Provide a fact about' is the key phrase, which we defined earlier and will elaborate on here. It means that for a given primary-key value, we can identify, at most, one of every data element that depends on that primary key. For example, assume **Department Number** is the primary key to the Department entity. Table 9.4 shows some sample values.

Table 9.4: Sample values for Department

Open table as spreadsheet

Department Number	Department Name
A	Accounting
A	Marketing
B	Information Technology
C	Manufacturing

In this example, **Department Number** A identifies two values for **Department Name**: Accounting and Marketing. Therefore, **Department Name** does not provide a fact about **Department Number**, and this example violates 1NF.

Ensuring each data element provides a fact about its primary key includes correcting the more blatant issue shown in Table 9.4, as well as addressing repeating groups and multi-valued data elements. Specifically, the modeler needs to:

- Move repeating data elements to a new entity
- Separate multi-valued data elements

Resolve Repeating Data Elements

When there are two or more of the same data element in the same entity, they are called repeating data elements. The reason repeating data elements violate 1NF is that for a given primary key value, we are getting more than one value back for the same data element.

Repeating data elements often take a sequence number as part of their name. For example, recall Figure 8.5, a subset reproduced here in Figure 9.8.

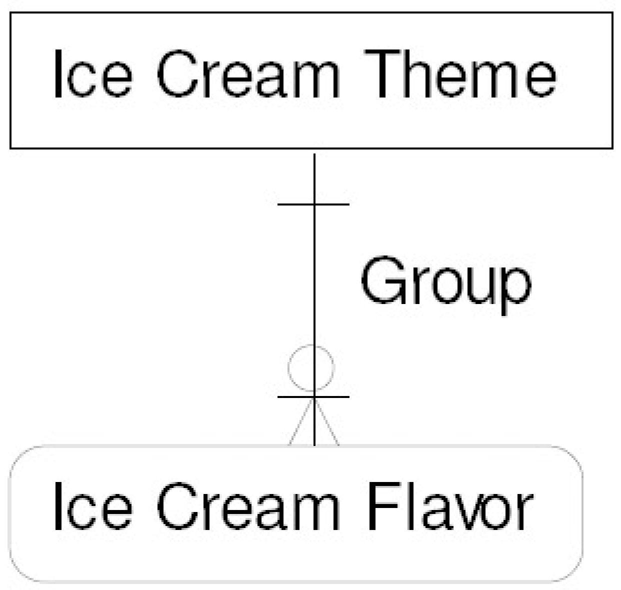


Figure 9.8: Ice Cream Theme can group many Ice Cream Flavors

Recall these definitions:

- **Ice Cream Flavor —** The distinctive taste of one ice cream product over another. Flavors are recognized by consumers. Examples include Chocolate, Vanilla, and Strawberry.
- **Ice Cream Theme —** A collection of ice cream flavors that have been grouped together because they share at least one common property. For example, the Brownie Ice Cream and Cake Batter Ice Cream flavors are grouped into the Cake Ice Cream Theme.

Imagine the initial logical model for this ice cream example is the model shown in Figure 9.9.

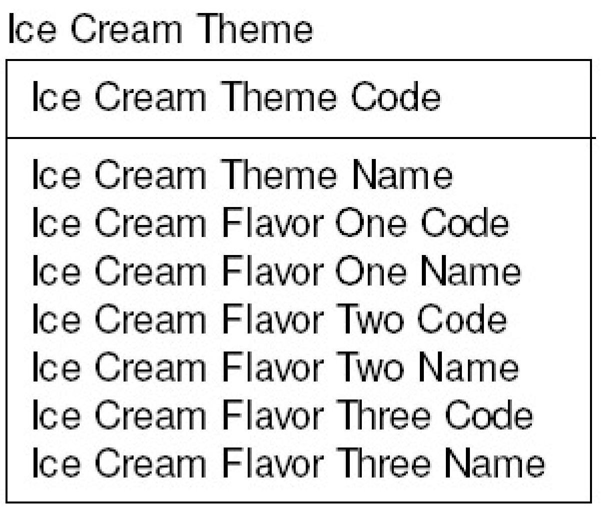

Ice Cream Theme
Ice Cream Theme Code
Ice Cream Theme Name
Ice Cream Flavor One Code
Ice Cream Flavor One Name
Ice Cream Flavor Two Code
Ice Cream Flavor Two Name
Ice Cream Flavor Three Code
Ice Cream Flavor Three Name

Figure 9.9: Un-normalized ice cream example

Table 9.5 contains the sample data for this model.

Table 9.5: Sample values for Ice Cream Theme

Open table as spreadsheet

Ice Cream Theme Code	**Ice Cream Theme Name**	**Ice Cream Flavor One Code**	**Ice Cream Flavor One Name**	**Ice Cream Flavor Two Code**	**Ice Cream Flavor Two Name**	**Ice Cream Flavor Three Code**	**Ice Cream Flavor Three Name**
C	Cake - Cake Ice Cream Theme	Br	Brownie	Ca	Cake Batter		
T	Trad — Traditional Ice Cream Theme	Ch	Chocolate	Va	Vanilla	St	Strawberry
W	Winter — Winter Ice Cream Theme	Sn	Snowball Slide				

Note that to resolve this repeating group, we need to ask important business questions such as:

- Is there any significance to the 'One', 'Two', or 'Three' in the name of Ice Cream Flavor? That is, could **Ice Cream Flavor One Name** be the most popular flavor, **Ice Cream Flavor Two Name** the second most popular, and **Ice Cream Flavor Three Name** the third most popular?
- Can you ever have more than three flavors?

- Can you ever have no flavors?

Repeating data elements hide business rules and limit the 'many' in a one-to-many or many-to-many relationship. If a data element is repeated three times in an entity, we can have at most three occurrences of this data element for a given entity instance. We cannot have four, and we waste space if we have only one or two.

Assume there is no significance to the 'One', 'Two', or 'Three' in the flavor name. Figure 9.10 contains the ice cream data model with repeating data elements resolved.

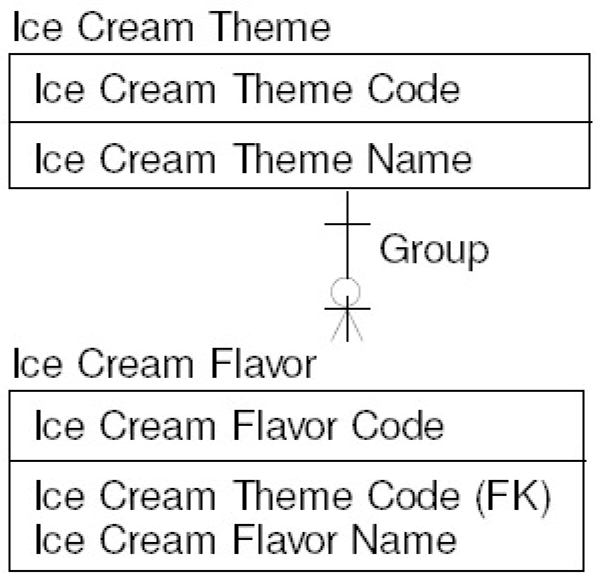


Figure 9.10: Ice Cream example with repeating data elements resolved

Now we can have any number of Ice Cream Flavors for a given Ice Cream Theme, such as zero, three, or one hundred.

In our contact example, by examining the sample data in Table 9.3 and asking lots of questions of business experts, we learn that a Contact can have more than one phone number and specialty description. We see that Jon Smith has three phone numbers: *(908)333-1212, (908)555-1212, 554-1212.* We see that The Amazing Rolando has three specialty descriptions: *Magic for all occasions, Walk around magic* and *Children's parties.*

We can find ourselves asking many questions just to determine if there are any repeating data elements we need to address. We can have a question template, such as:

- "Can a" <<insert entity name here>> "have more than one" <<insert data element name here>> "?"

So these are all valid questions:

- Can a Contact have more than one **Email Address**?

- Can a Contact have more than one **Person Name**?
- Can a Contact have more than one **Person Title**?
- Can a Contact have more than one **Company Name**?
- Can a Contact have more than one **Web Address**?
- Can a Contact have more than one **Mailing Address**?
- Can a Contact have more than one **Phone Number**?
- Can a Contact have more than one **Logo Image**?
- Can a Contact have more than one **Specialty Description**?

Note that you might find yourself rephrasing some of these techie-sounding questions to make them more understandable to a business person. "Can a Contact have more than one **Email Address**?" might be better phrased as "Do you ever have the need to reach a Contact by more than one **Email Address**?"

Assuming that the four business cards we are examining contain a good representative set of data (and that's a big assumption), and that the businesspeople provide answers to our questions that are consistent with this data, a more accurate version of our Contact model would reflect that a contact can have at most three phone numbers and at most three specialties, as shown in Figure 9.11.

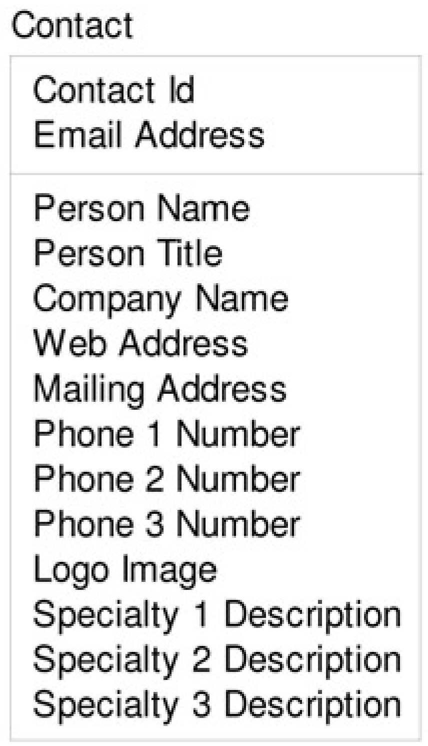


Figure 9.11: More accurate view of Contact

If a Contact can have three phone numbers, there is a good chance that one day there will be a Contact that has four phone numbers, or maybe ten. So in general terms, a Contact can have one or many phone numbers, and also one or many specialties. So we need to create separate entities for phone number and specialty, as shown in Figure 9.12.

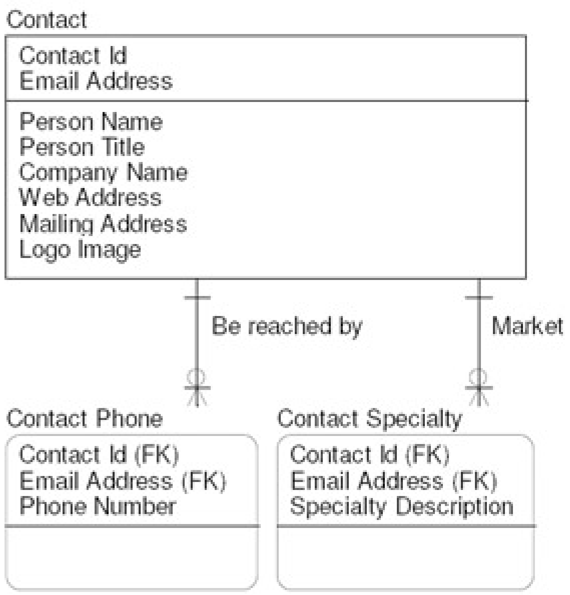


Figure 9.12: Repeating groups moved to new entities

To resolve a repeating data element, you can see that we need to create a one-to-many relationship or a many-to-many relationship with a new entity that contains the repeating data element.

We are not yet done with completely resolving repeating groups, though. What other questions do we need to ask?

We already asked whether a contact can have more than one of each of its data elements. We also need to ask the other side of the equation, which is - can the values in these repeating data elements belong to more than one contact?

We need to make sure a one-to-many relationship from Contact to Contact Phone, and from Contact to Contact Specialty is correct, and that it is not a many-to-many relationship. So these two business questions need to be answered, as well:

- Can the same **Phone Number** belong to more than one Contact?
- Can the same **Specialty Description** belong to more than one Contact?

By looking at the sample data in Table 9.3 and confirming with businesspeople, we learn that the same phone number and specialty can belong to more than one contact, as Jenn and Steve from findsonline.com both share the phone number *(973)555-1212* and specialty *Internet auction experts*. Therefore, a more accurate model with repeating groups resolved is shown in Figure 9.13.

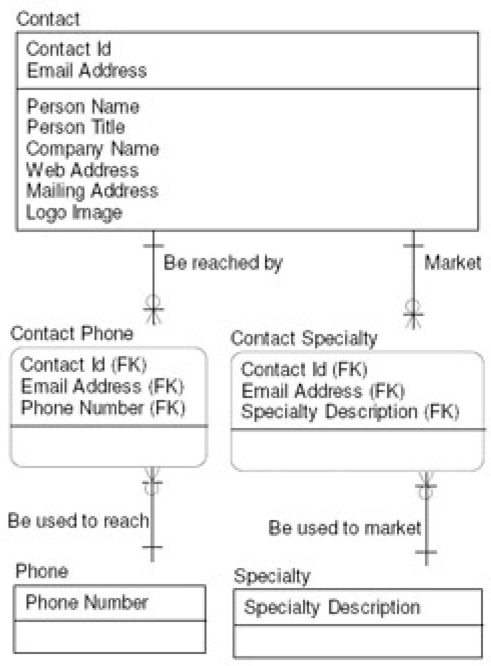


Figure 9.13: Repeating groups resolved

Note that what currently makes the phone number and specialty description unique at this point, is the **Phone Number** and **Specialty Description** themselves, respectively. These are not ideal primary keys. but they serve to illustrate normalization in this example.

Resolve Multi-valued Data Elements

Multi-valued means that within the same data element we are storing at least two distinct values. There are at least two different business concepts hiding in one data element. For example, Name may contain both a first name and last name. **First Name** and **Last Name** can be considered distinct data elements, and therefore, "John Smith" stored in Name, is multi-valued, because it contains both "John" and "Smith."

Using our ice cream example from Figure 9.9 with sample values in Table 9.5, recall the values for **Ice Cream Theme Name**:

- Cake - Cake Ice Cream Theme
- Trad — Traditional Ice Cream Theme
- Winter — Winter Ice Cream Theme

By examining this sample data and then validating it through a business expert, we see that **Ice Cream Theme Name** contains more than just the name of the theme. It also contains a theme description. Therefore, to resolve repeating data elements, we need to break description out from name, as shown in Figure 9.14. This now puts our data model in 1NF.

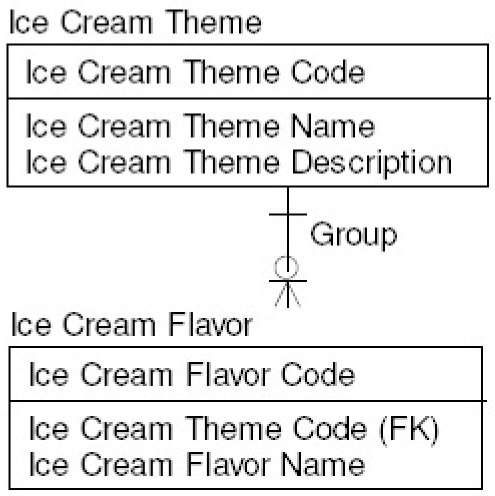


Figure 9.14: Ice cream example in 1NF

Returning to our contact example, by examining the data in Table 9.3 and the data model in Figure 9.13, and by asking even more questions, we can identify those data elements that need to be broken up into more refined data elements.

We may find ourselves asking many questions just to determine if there are any multi-valued data elements we need to identify. We can have another question template, such as:

- "Does a" <<insert data element name here>> "contain more than one piece of business information?"

So these are all valid questions:

- Does a **Contact Id** contain more than one piece of business information?
- Does an **Email Address** contain more than one piece of business information?
- Does a **Person Name** contain more than one piece of business information?
- Does a **Person Titl**e contain more than one piece of business information?
- Does a **Company Name** contain more than one piece of business information?
- Does a **Mailing Address** contain more than one piece of business information?
- Does a **Logo Image** contain more than one piece of business information?
- Does a **Web Address** contain more than one piece of business information?
- Does a **Phone Number** contain more than one piece of business information?
- Does a **Specialty Description** contain more than one piece of business information?

Often a modeler will encounter multi-valued data elements that do not need to be separated into distinct data elements. This is common when the distinct data element parts are not of interest to the business or industry and the data element is considered to be in its atomic form, even though it contains more than one piece of information. For example, **Phone Number** contains an area code and may contain a country code. We might, therefore, decide to show country code and area code as separate data elements on the model. However, do your businesspeople ever need to see the area code or do they view phone number in the most granular form? The answer to this question determines whether the modeler breaks apart the **Phone Number** data element.

The cost of breaking apart a multi-valued data element unnecessarily is the potential for confusion when explaining the data model, extra development effort to break apart the actual data, and then extra development effort to put the pieces back together for the business user. There is also a chance that in putting the pieces back together, some data may be lost or changed and the newly combined values no longer match the original data, thereby causing a data quality issue. Looking at the values from Table 9.3, we see that **Phone Number** contains different formats, and one phone number is even missing an area code. Phone Number, therefore, would be an example of a data element that would require extra development effort to break apart.

Assume that in our contact example, **Person Name** and **Mailing Address** are the only two data elements that require being shown in a more granular form. The model in Figure 9.15 resolves these two multi-valued data elements and is therefore in 1NF.

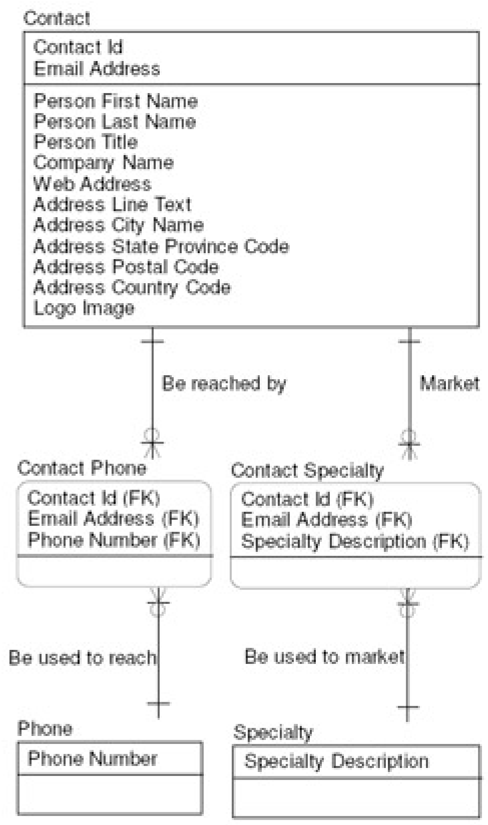


Figure 9.15: Contact data model with multi-valued data elements resolved, and therefore in 1NF

Second Normal Form (2NF)

Recall the summary of all three normalization levels: *Every data element must provide a fact about the key*, *the whole key*, *and nothing but the key*. First Normal Form (1NF) is the "Every data element must provide a fact about the key" part. Second Normal Form (2NF) is "the whole key" part. This means each entity must have the minimal set of data elements that uniquely identifies each entity instance.

For example, in Figure 9.16, we have Employee Assignment.

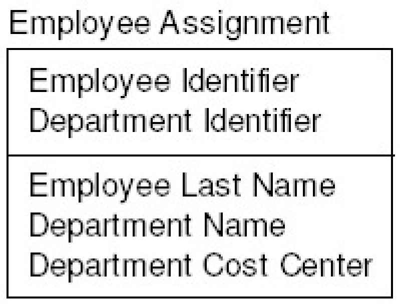


Figure 9.16: Example of model not in 2NF

The primary key of Employee Assignment is **Employee Identifier** and **Department Identifier**. Do we have the minimal primary key for each of the non-key data elements? We would need to confirm with business experts whether the **Employee Identifier** is unique within a department or unique across the whole company. Most likely it is unique across the whole company, so we therefore do not have the minimal primary key for each of the non-key data elements. **Employee Last Name** probably needs only **Employee Identifier** to retrieve one value. Similarly, both **Department Name** and **Department Cost Center** most likely need only **Department Identifier** to retrieve one of their values. Therefore, we need to modify this model to get it into 2NF, as shown in Figure 9.17.

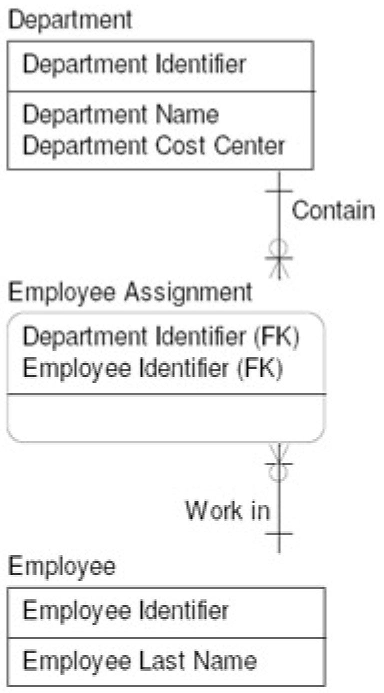


Figure 9.17: One option for putting the model in 2NF

Now **Department Name** and **Department Cost Center** are facts about only **Department Identifier**, and **Employee Last Name** is a fact about only **Employee Identifier**. The associative entity Employee Assignment links employees with their departments.

Normalization is a process of asking business questions. In this example, we could not complete 2NF without asking the business "Can an Employee work for more than one Department?" If the answer is "Yes" or "Sometimes", then the model in Figure 9.17 is accurate. If the answer is "No", then the model in Figure 9.18 prevails.

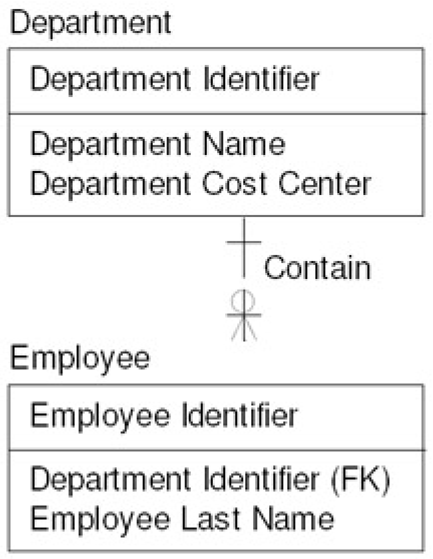


Figure 9.18: Another option for putting the model in 2NF

This model assumes answers to two other business questions:

- Can a Department exist without an Employee?
- Can an Employee exist without a Department?

The answers from the business would be "Yes, a Department can exist without an Employee" and "No, an Employee cannot exist without a Department".

As with 1NF, we will find ourselves asking many questions to determine if we have the minimal primary key. We can have another question template, such as:

- "Are all of the data elements in the primary key needed to retrieve a single instance of" <<insert data element name here>>?" In our Contact example, the 'minimal set of primary key instances' are **Contact Id** and **Email Address**.

So these are all valid questions for our contact example:

- Are both Contact Id and Email Address needed to retrieve a single instance of **Person First Name**?

- Are both Contact Id and Email Address needed to retrieve a single instance of **Person Last Name**?
- Are both Contact Id and Email Address needed to retrieve a single instance of **Person Title**?
- Are both Contact Id and Email Address needed to retrieve a single instance of **Company Name**?
- Are both Contact Id and Email Address needed to retrieve a single instance of **Web Address**?
- Are both Contact Id and Email Address needed to retrieve a single instance of **Address Line Text**?
- Are both Contact Id and Email Address needed to retrieve a single instance of **Address City Name**?
- Are both Contact Id and Email Address needed to retrieve a single instance of **Address State Province Code**?
- Are both Contact Id and Email Address needed to retrieve a single instance of **Address Postal Code**?
- Are both Contact Id and Email Address needed to retrieve a single instance of **Address Country Code**?
- Are both Contact Id and Email Address needed to retrieve a single instance of **Logo Image**?

We realize the answer to all of these questions is "No". We do not need both the **Contact Id** and **Email Address** in the primary key of Contact. Either one of these is enough to uniquely identify a contact and return a single instance of any of the data elements in Contact. Therefore, we only need one of these in the primary key. Let's use **Contact Id** as the primary key, and make **Email Address** an alternate key. The updated model is shown in Figure 9.19.

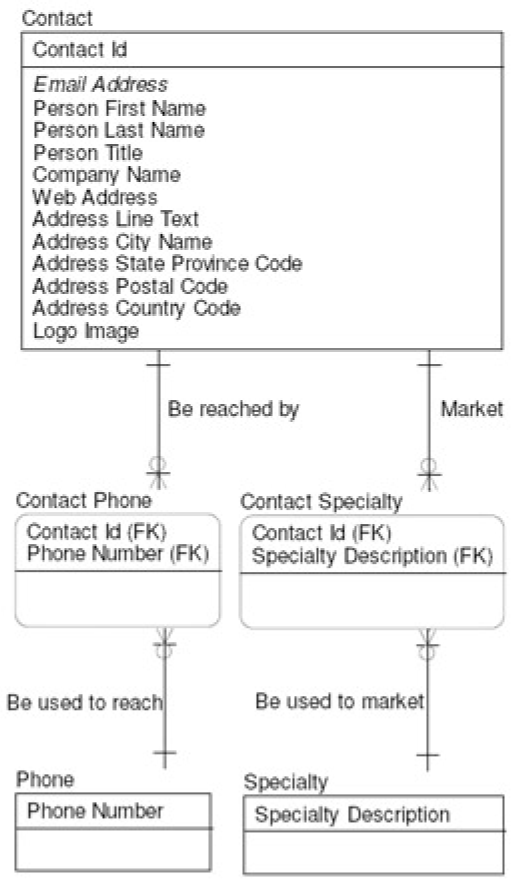


Figure 9.19: Contact model in 2NF

Asking the previous list of questions at this point might lead us to a more descriptive and refined model. For example, we realize there are people and company information in this entity. We can therefore make our model more descriptive and capture more business rules if we introduce subtyping into this structure.

Subtyping captures many more rules, but before we can show them, we need to confirm them with the business by asking more questions. For example, here are the questions we would need to ask to arrive at the model in Figure 9.20:

- Which data elements in Contact are shared by both a Person and Company?
- Which data elements are just for a Person?
- Which data elements are just for a Company?
- Which relationships to Contact are shared by both a Person and Company?

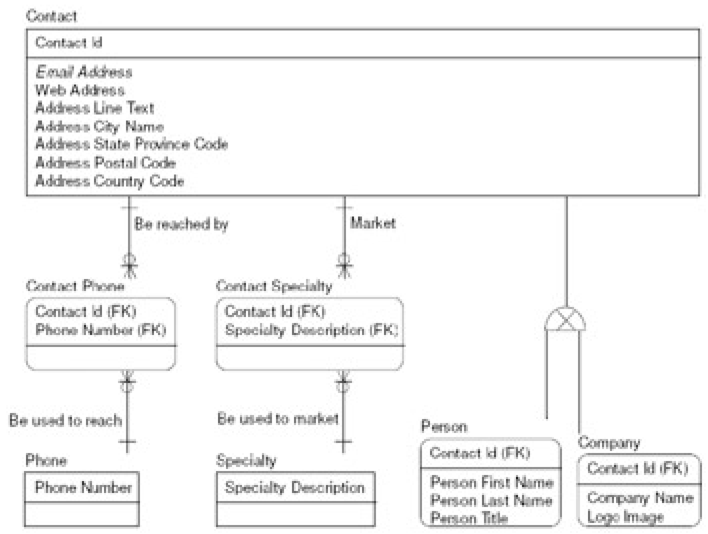


Figure 9.20: Contact model in 2NF updated with subtyping

Note that we learn that the Logo Image is really just for a Company and not for a Person.

In a subtyping structure, the primary key from the supertype is also the primary key for each subtype. By default, the primary key for the subtype has the same name as the primary key for the supertype, as shown in Figure 9.20. The modeler often renames the subtype primary keys to make them more meaningful. Renaming any foreign key, including a subtype's primary key, is called role naming. For example, we can role name **Contact Id** in Person to **Person Id** and **Contact Id** in Company to **Company Id. Person Id** and **Company Id** still contain the same value as **Contact Id**, but renaming the subtype's primary key facilitates better communication. Figure 9.21 contains the model with the new names.

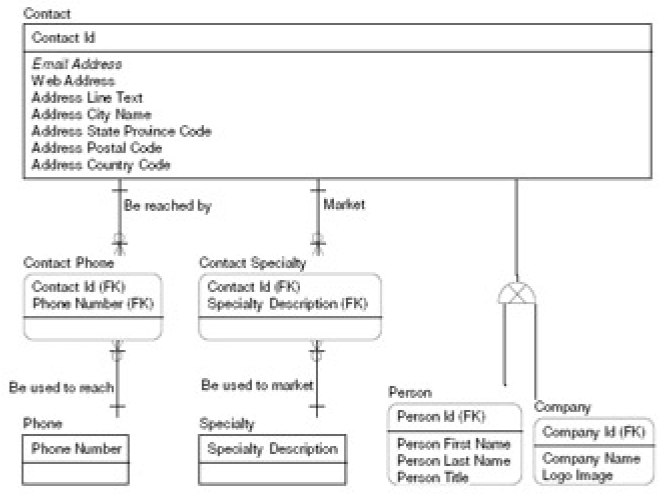


Figure 9.21: Contact model in 2NF updated with role names

An important relationship that was assumed in Figure 9.19 is missing here. People work for Companies. By having the data elements together in the same entity in Figure 9.19, we assumed this was the case. Now, we have to show it explicitly in Figure 9.21. Can a Person work for more than one Company? Can a Company contain more than one Person? These questions need to be answered by the business.

Let's assume the answers to these questions are that a Person can only work for one Company, but doesn't have to work for a Company at all, and that a Company can employ many People, but doesn't have to employ any People. The updated model is shown in Figure 9.22.

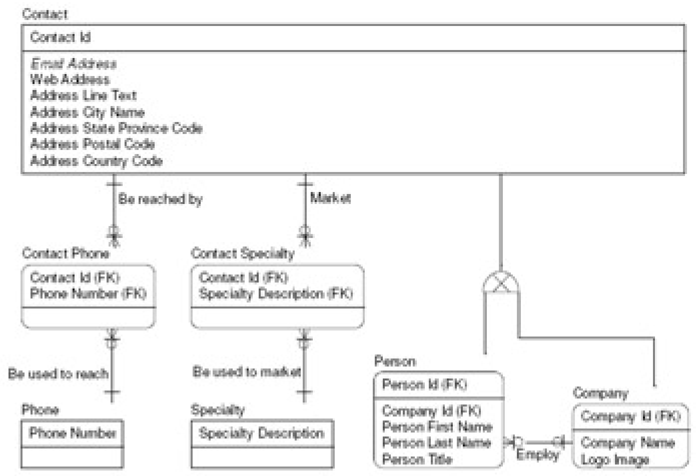


Figure 9.22: Contact model in 2NF updated with Person Company relationship

Third Normal Form (3NF)

Again, recall our summary of all three normalization levels: *Every data element must provide a fact about the key, the whole key, and nothing but the key*. First Normal Form (1NF) is the "Every data element must provide a fact about the key" part. Second Normal Form (2NF) is "the whole key" part, and Third Normal Form (3NF) is the "nothing but the key" part.

3NF requires the removal of hidden dependencies. Each data element must be directly dependent on the primary key, and not directly dependent on any other data elements within the same entity. Alternate and foreign keys are excluded from this test.

The data model is a communication tool. The relational logical data model communicates which data elements are facts about the primary key and only the primary key. Hidden dependencies complicate the model and make it difficult to determine how to retrieve values for each data element.

To resolve a hidden dependency, you will either need to remove the data element that is a fact about non-primary key data element(s) from the model, or you will need to create a new entity with a different primary key for the data element that is dependent on the non-primary key data element(s).

As with 1NF and 2NF, we will find ourselves asking many questions to uncover hidden dependencies. We can have another question template, such as:

- "Is" <<insert data element name here>> "a fact about any other data element in this same entity?"

So these are all valid questions for the Contact entity within our contact example:

- Is **Web Address** a fact about any other data element within this same entity?
- Is **Address Line Text** a fact about any other data element within this same entity?
- Is **Address City Name** a fact about any other data element within this same entity?
- Is **Address State Province Code** a fact about any other data element within this same entity?
- Is **Address Postal Code** a fact about any other data element within this same entity?
- Is **Address Country Code** a fact about any other data element within this same entity?

We learn from asking these questions that there are some hidden dependencies within the address data elements. If we know the postal code, we can determine the city, state province, and country. Therefore, by moving city, state province, and country to a different entity, as shown in Figure 9.23 our model is now in 3NF.

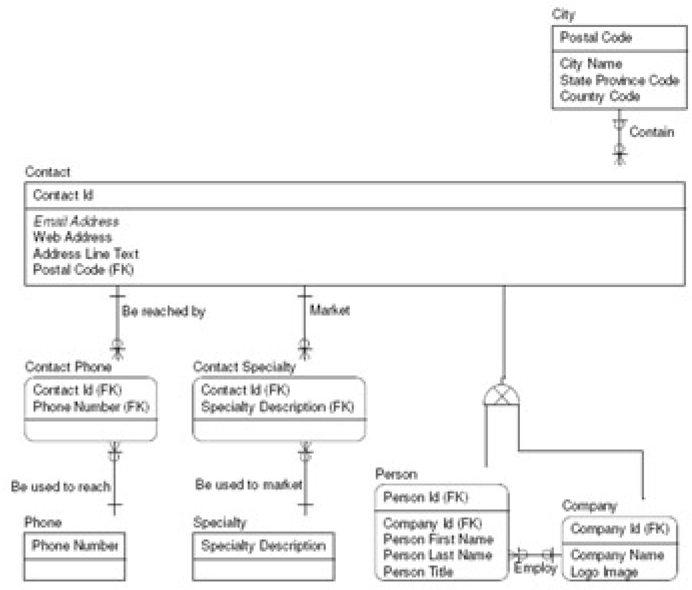


Figure 9.23: Contact model in 3NF

You may be wondering why we did not also move country and state province to their own entities. After all, if we know the city, we should know the state province, and if we know the state province, we should know the country. The reason we did not break these apart is because we do not have enough information to guarantee we can find the single country for a given state province or the state province for a given city. There is a Long Beach in New York State and a Long Beach in California, for example.

Note that we needed to ask even more questions to determine the right cardinality on the relationship line between City and Contact:

- Does a Contact have to have a City?
- Does a City have to contain at least one Contact?

We learned from the business that no, a contact does not have to have a city. We also learned that there could be cities in which no contacts reside. Therefore, the answer to the second question is "No".

You will find that the more you normalize, the more you go from applying rules sequentially to applying them in parallel. For example, instead of first applying 1NF to your model everywhere, and then when you are done applying 2NF, and so on, you will find yourself looking to apply all levels at once. This can be done by looking at each entity and making sure the primary key is correct and that it contains a minimal set of data elements, and that all data elements are facts about only the primary key.

Abstraction Explained

Normalization is a mandatory technique on the relational logical data model. Abstraction is an optional technique. As mentioned earlier, abstraction brings flexibility to your data models by redefining and combining some of the data elements, entities, and relationships within the model into more generic terms.

Returning to our contact example, we may decide to abstract Email Address and the phone number structure into a more generic Communication Medium structure. Communication Medium could include any method of communicating with a contact, including email and phone numbers, as well as all future types of communication mediums such as text messaging or anything else that comes along. This more abstract contact data model, based on the model from Figure 9.23, is shown in Figure 9.24. Table 9.6 contains some of the values in the Communication Medium structure.

Table 9.6: Business card values in Communication Medium structure

Open table as spreadsheet

Contact			
Contact Id	**Web Address**	**Address Line Text**	**Postal Code**
123	www.stevehoberman.com	10 Main St	10021
54	findsonline.com		
58	findsonline.com		
42			
14		58 Church Avenue	08901

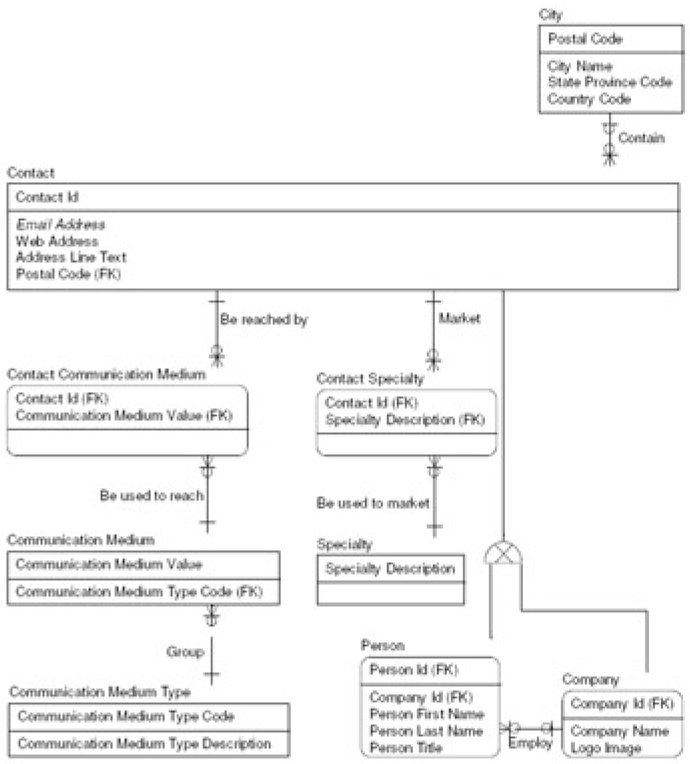


Figure 9.24: Contact data model with C ommunication Medium

Open table as spreadsheet

Communication Medium Type	
Communication Medium Type Code	**Communication Medium Type Description**
01	Telephone
02	Fax Number
03	Email Address
04	Text Messaging

Open table as spreadsheet

Communication Medium	
Communication Medium Value	**Communication Medium Type Code**
212-555-1212	01
(973)555-1212	01
732-555-1212	01
908-333-1212	01
908-555-1212	02
554-1212	01
me@stevehoberman.com	03

Communication Medium	
Communication Medium Value	**Communication Medium Type Code**
Steve@findsonline.com	03
Jenn@findsonline.com	03
BillSmith@TheAmazingRolando.com	03
Reservations@RaritanRiverClub.com	03

Open table as spreadsheet

Contact Communication Medium	
Contact Id	**Communication Medium Value**
123	212-555-1212
54	973-555-1212
58	973-555-1212
42	732-555-1212
14	908-333-1212
14	908-555-1212
14	554-1212
123	me@stevehoberman.com
54	Steve@findsonline.com
58	Jenn@findsonline.com
42	BillSmith@TheAmazingRolando.com
14	Reservations@RaritanRiverClub.com

Notice the extra flexibility we gain with abstraction. A new email address for findsonline.com, for example, leads to a new entity instance in Communication Medium and Contact Communication Medium instead of a new data element on a model, with subsequent database and application changes. Abstraction allows for greater flexibility, but does come with a price. Actually, three high prices:

- **Loss of communication.** The concepts we abstract are no longer represented explicitly on the model. That is, when we abstract, we often convert column names to entity instances. For example, **Email Address** is no longer a data element on the data model in Figure 9.24, but is, instead, an entity instance of Communication Medium, with a **Communication Medium Type Code** value of '03' for 'Email Address'. One of the main reasons we model is for communication, and abstracting can definitely hinder communication.
- **Loss of business rules.** When we abstract, we can also lose business rules. To be more specific, the rules we enforced on the data model before abstraction now need to be enforced through other means, such as through programming code. If we wanted to

enforce that a Contact must have one and only one **Email Address,** for example, we can no longer enforce this rule through the abstracted data model in Figure 9.24.

- **Additional development complexity.** Abstracting requires sophisticated development techniques to turn columns into rows when loading an abstract structure, or to turn rows back into columns when populating a structure from an abstract source. Imagine the work to populate the data elements in Communication Medium from a source data element called **Email** that only contains the email address. It would be much easier for a developer to load a data element called **Email** into a data element called **Email Address**. The code would be simple and it would be very fast to load.

So, although abstraction provides flexibility to an application, it does come with a cost. It makes the most sense to use abstraction when the modeler or analyst anticipates additional types of something coming in the near future. For example, additional types of communication mediums such as text messaging would be a good justification for using the Communication Medium structure in Figure 9.24. Make sure you only abstract where it makes sense.

Dimensional Modeling FAQ

In a previous section of this chapter, we discussed the dimensional logical data model. There are quite a few terms and guidelines in dimensional modeling that differ from relational modeling. This section covers these terms and guidelines in the form of Frequently Asked Questions (FAQ).

Recall the dimensional model from Figure 9.3, repeated below as Figure 9.25, as we go through each of these questions.

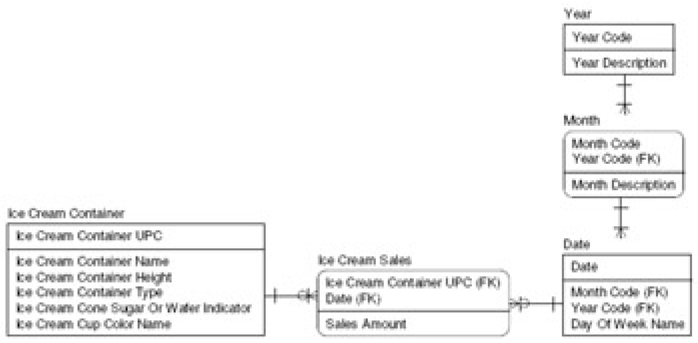


Figure 9.25: Dimensional logical data model of ice cream

Should you always model the lowest level of detail available in the business? The lowest level of detail for the meter is known as the grain. The grain in Figure 9.25, for example, is Ice Cream Container and Date. Do you design exactly for the questions the business needs answered, or do you add a more detailed grain in anticipation of more detailed questions with the same measures? My belief is that it is ok to provide the richer grain as long as it does not impact performance (e.g. Will it take a lot longer to return results at a date level instead of at a month level?), scope-creep (e.g. Will the project take another month to finish because of the extra grain?), data quality (e.g. Will the business be responsible for checking the quality of additional data?), or user-friendliness (Is the dimensional model no longer easy to understand?).

Can you relate dimensions to each other? Not on a dimensional logical data model. All relationships from dimensions must go through the meter. That is, you can never have relationships between different dimensions. In the dimensional model from Figure 9.25, you cannot show, for example, the rule capturing which Ice Cream Containers are available in a given Year. That is, we cannot create a relationship between Ice Cream Container and Year.

Can you have optional cardinality on the relationship lines connecting dimension levels? No, don't leave navigation paths empty. It is a good practice to avoid null (i.e. empty) foreign keys on a dimensional model, because nulls can be interpreted differently by different databases and reporting tools (and users!). So to avoid unnecessary confusion and data discrepancies, make sure every hierarchy level is populated. If a specific Product does not roll up to a specific Product Line for example, using a default value instead of leaving the foreign key in Product empty will minimize reporting errors and misunderstandings.

Can you combine relational and dimensional modeling techniques on the same logical model? No, not on the same logical model. A logical model must be relational OR dimensional; it cannot have properties of both. When you get to the physical model, one solution would be to combine them within the same schema. There are pros and cons, though (like everything in the physical). Pros include meeting multiple needs within the same structure, while cons include creating reporting complexity such as loops. (A loop is when there is more than one navigation path to arrive at the meter.)

What is a conformed dimension? A conformed dimension is one that is shared across the business intelligence environment. Customer, Account, Employee, Product, Time, and Geography are examples of conformed dimensions. The term was made popular by Ralph Kimball and requires the modeler to design the conformed dimension with a much broader perspective than just the requirements for a single data mart. Conformed dimensions allow the navigator to ask questions that cross multiple marts. For example, a user could navigate from a Promotions data mart directly to a Sales data mart to track the impact of certain promotions. Most dimensions need to be conforming. In fact, it is rare to come across dimensions that will only ever be relevant for a single data mart.

What is a factless fact? The meter on a dimensional logical data model, is called a *fact table* on a dimensional physical data model. A fact table that does not contain any facts (i.e. measures) is called a factless fact (good name, huh?). Factless facts count events by summing relationship occurrences between the dimensions. For example, a fact table called Attendance contains no measures. It links to the Student, Course, and Semester dimensions with the goal of counting relationship occurrences of how many students take a particular course in a particular semester.

Exercise 9: Modifying a Logical Data Model

1. You are working on a subset of a logical data model, as shown in Figure 9.26.

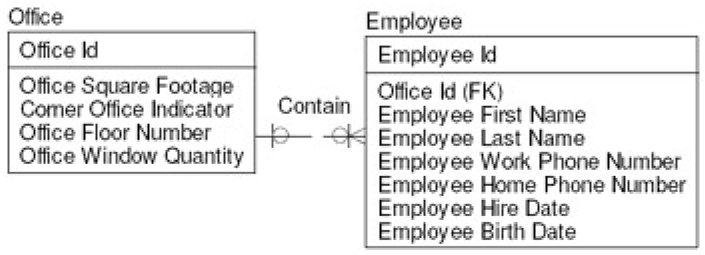


Figure 9.26: LDM subset

The relationship states:

- Each Office may contain one or many Employees.
- Each Employee may reside in one Office.

You need to add the data element **Office First Occupied Date** to this model.

Here is the definition of **Office First Occupied Date:**

- ***Office First Occupied Date*** *is the official date, as recorded in the Human Resources database, when a person first took up residence in an office. It must be a business day (that is, not a weekend day or holiday).*

The business expert in this area provides you with the following three rules that need to be captured on this model:

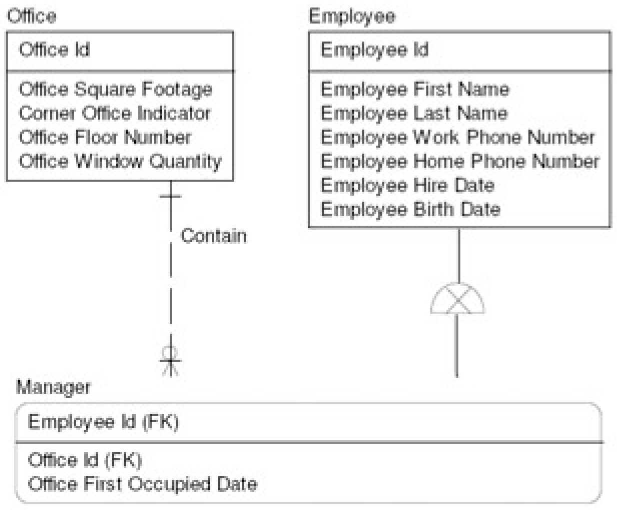
1. **Office First Occupied Date** is a mandatory field for each employee who has an office.
2. Only managers have an office and all managers have an office.
3. A manager has one and only one office.

Modify this model to accommodate this new data element and these 3 business rules.

Answers

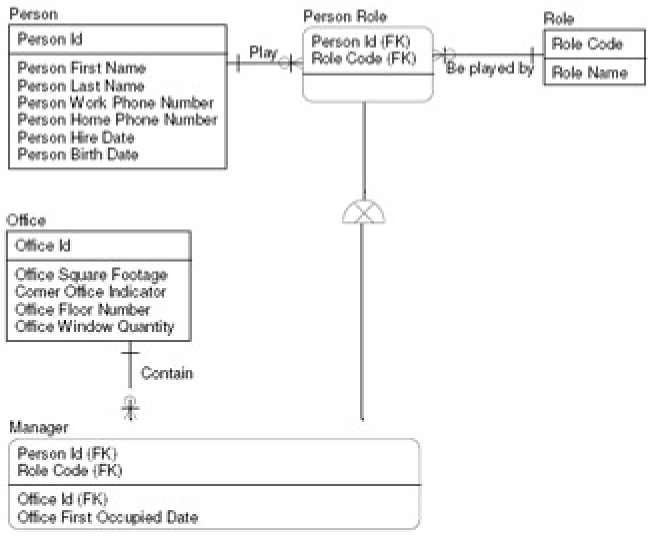
1. There are two ways to model this situation. The key point with this challenge is to use subtyping and keep Office First Occupied Date in the subtype as not null.

OPTION 1



In this model we subtyped Manager as a type of Employee. All of the data elements and relationships that are specific to a Manager are moved to the Manager entity. Therefore, the business rule that each Manager must reside in one and only one Office and each Office may contain one or many Managers is represented. Also, the Office First Occupied Date is now a mandatory field in the Manager entity, whereas in the Employee entity it would have to be null.

OPTION 2 (A BIT MORE ABSTRACT)



Thinking in more generic terms, we might model that a Person can play many Roles. One of these Roles is a Manager. The Manager subtype has the additional relationship and mandatory Office First Occupied Date that we saw in the previous model. This type of model works out well where application longevity and stability is the goal, such as in a data warehouse or integration hub.

See the Appendix for my answers.

Key Points

- A logical data model (LDM) represents a detailed business solution.
- A relational logical model represents how the business works. A dimensional logical model represents what the business is monitoring.
- Normalizing is a formal process of asking business questions. Normalization ensures that every data element is a fact about the key (1NF), the whole key (2NF), and nothing but the key (3NF).
- Abstraction brings flexibility to your logical data models by redefining and combining some of the data elements, entities, and relationships within the model into more generic terms.
- There are a number of important terms unique to dimensional modeling, including factless facts and conforming dimensions.
- Dimensional modeling requires having mandatory cardinality on relationship lines and not relating dimensions to each other.

Chapter 10: What are Physical Data Models?

Overview

Let's get Physical
Consider environment
Time to make it real

The highlighted row in Table 10.1 shows the focus of this chapter, which is the physical data model.

Table 10.1: The PDM is the focus of this chapter

⇨Open table as spreadsheet

	Relational	**Dimensional**
SAM	'One-pager' on how something works	'One-pager' on what is monitored
LDM	Detailed business solution on how something works	Detailed business solution on what is monitored
PDM	**Detailed technical solution on how something works**	**Detailed technical solution on what is monitored**

A physical data model (PDM) takes the business solution defined on a logical data model to the next level of a technical solution. That is, once you solve the problem independent of software and hardware concerns, then you can make adjustments for software and hardware. This chapter will explain the most popular techniques used to make adjustments to a business solution to create an efficient technical solution. I will explain the PDM and then discuss the techniques of denormalization, views, indexing, and partitioning. Although these techniques apply to both relational and dimensional models, their names may differ depending on which type of model they are applied to. I will explain these terminology differences in this chapter, as well. I will conclude with a discussion on how to adjust your physical data model to accommodate data value changes, and introduce the concept of a slowly changing dimension.

Physical Data Model Explained

The physical data model (PDM) is the logical data model modified for a specific set of software or hardware. Recall the camera analogy discussed earlier. The logical data model is analogous to the negative which, although a perfect view of the subject matter, does not display well in a picture frame or in a photo album. The photograph is an instantiation of the negative. Similarly, the PDM is an instantiation of the LDM. A PDM may not have all of the accuracy and rules found on a LDM, just as the photograph lacks the perfection of the negative. The PDM often gives up perfection for practicality, factoring in real concerns such as speed, space, and security.

On the SAM, we might learn, for example, what the terms, business rules, and scope would be for a new order entry system. After understanding the need for an order entry system, we create a LDM representing the business solution. It contains all of the data elements and business rules needed to deliver the system. For example, the subject area model will show that a Customer places many Orders. The LDM will capture all of the details behind Customer and Order, such as the customer's name, their address, and the order number. After understanding the business solution, we move on to the technical solution and build the PDM.

While building the PDM, we address the issues that have to do with specific hardware or software such as:

- How can we retrieve this information in fewer than 5 seconds?
- How can we make this information secure?
- There is a lot of information here. What is the best way to manage storage space?

Note that in the early days of data modeling, when storage space was expensive and computers were slow, there were major modifications made to the PDM to make it work. In some cases, the PDM looked like it was for an entirely different application than the LDM. As technology improved, the PDM started looking more like the LDM. We should continue to see this trend as better hardware and software lead to fewer compromises on the PDM, resulting in a PDM that looks more like its LDM.

Denormalization Explained

Denormalization is the process of selectively violating normalization rules and reintroducing redundancy into the model (and therefore, the database). This extra redundancy can reduce data retrieval time, which is the primary reason for denormalizing. We can also denormalize to create a more user-friendly model. For example, we might decide to denormalize company information into an entity containing employee information, because usually when employee information is retrieved, company information is also retrieved.

The faster retrieval and user-friendliness, however, come with the price of extra redundancy on the model, which could in turn:

- **Cause update, delete, and insert performance to suffer.** When we denormalize, we often repeat the value of one or more data elements, which helps to reduce retrieval time. However, if we have to change the value that we are repeating, we need to change it everywhere it occurs. If we are repeating Company information for each Employee, for example, and a **Company Name** value is changed, we will need to make this update for each Employee instance that works for this Company. Updating a value for **Company Name** would need to be done in multiple places, which takes more time than if **Company Name** appeared in just one place.
- **Introduce data quality problems.** By having the same value appear multiple times, we substantially increase opportunities for data quality issues when those values change. Imagine, for example, if we failed to update all Employee instances of a company name change if company information were denormalized into Employee. Denormalizing also reduces the number of relationships on the model, thereby reducing the number of business rules enforced on our model. Denormalizing company information into Employee means we can no longer enforce the rule that each Employee must work for a valid Company.
- **Take up more space.** In a table with a small number of records, the extra storage space incurred by denormalization is usually not substantial. However, in tables with millions of rows, every character could require megabytes of additional space. It is true that space is becoming cheaper every day. However, I recently had to purchase more storage space for one of my projects, and I was amazed at the amount of time it took to go through the budgeting process and red tape to actually receive the storage space. Space was cheap, but time was not. And depending on your specific hardware platform, space may not be that cheap.
- **Stunt growth of the application.** When we denormalize, it can become harder to enhance structures to meet new requirements. Before we add data elements or relationships, we need to understand all the rules that were explicitly shown through relationships on the logical data model and are now hidden on the physical data model. If Year and Month are denormalized into Date, we can quickly retrieve year and month information when viewing a particular date. However, it would be very challenging to expand this denormalized calendar structure to add Week, for example. We would have to sort through all the redundant data and then add new redundant data for Week, after understanding all of the business rules about how Dates roll up to Weeks and Weeks roll up to Months and Years.

There are five denormalization techniques:

- Standard

- FUBES
- Repeating groups
- Repeating data elements
- Summarization

We will apply each of these five techniques to the contact logical data model from the previous chapter, repeated here in Figure 10.1.

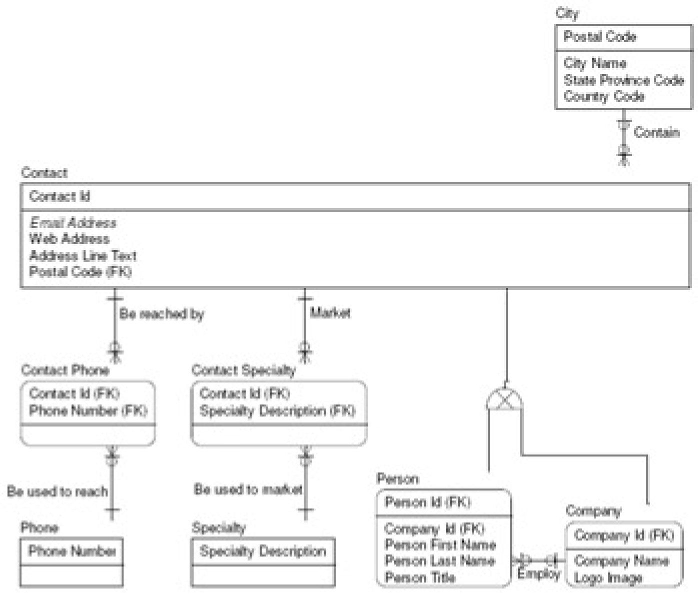


Figure 10.1: Contact logical data model

Standard

The standard method is the most common of the five denormalization techniques. The parent entity in the relationship disappears, and all the parent's data elements and relationships are moved down to the child entity. You'll recall that the child entity is on the many side of the relationship and contains a foreign key back to the parent entity, which appears on the one side of the relationship.

Although the standard technique is traditionally applied to a one-to-many relationship, we can illustrate the standard technique using the subtyping structure from Figure 10.1. The subtyping structure must be resolved on the physical data model, as there is no subtyping relationship defined in a relational database. There are three ways of resolving the subtyping symbol on the physical data model: identity, rolling down, and rolling up.

Identity is the closest to subtyping, itself, because the subtyping symbol is replaced with a one-to-one relationship for each supertype/subtype combination. The main advantage of identity is that all of the business rules at the supertype level and at the subtype level remain as in the logical. That is, we can continue to enforce relationships at the supertype or subtype levels, as well as enforce that certain data elements be required at the supertype or subtype levels. The main disadvantage of identity is that it can take more time to retrieve data, as it requires navigating multiple tables to access both the supertype and subtype information. Identity is not a form of denormalization, but it is shown in Figure 10.2 for completeness.

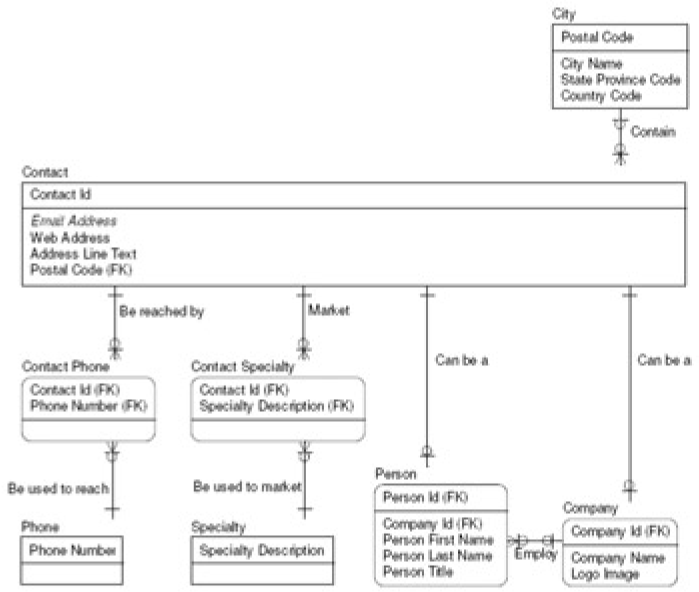


Figure 10.2: Identity method of resolving subtyping

In this example, we can continue to enforce certain rules at the supertype level, such as a Contact can have only one City and must have one **Email Address**. We can also enforce rules at the subtype level, such as a Person can work for one Company, and **Person Last Name** is a required data element and must always contain a value.

Rolling down is when the supertype is 'rolled down' into each subtype, moving all of the data elements and relationships from the supertype into each subtype, then removing the supertype from the data model. Rolling down can produce a more user-friendly structure than identity or rolling up, because subtypes are often more concrete concepts than supertypes, making it easier for the users of the data model to relate to the subtypes. For example, users of our contact model would probably feel more comfortable with the concepts of Person and Company, rather than the concept of Contact. However, we are repeating relationships and data elements, which could reduce any user-friendliness gained from removing the supertype. In addition, the rolling down technique enforces only those rules present in the subtypes. This could lead to a less flexible data model, as we can no longer easily accommodate new subtypes without modifying the data model. See Figure 10.3 for what rolling down would look like in our Contact example.

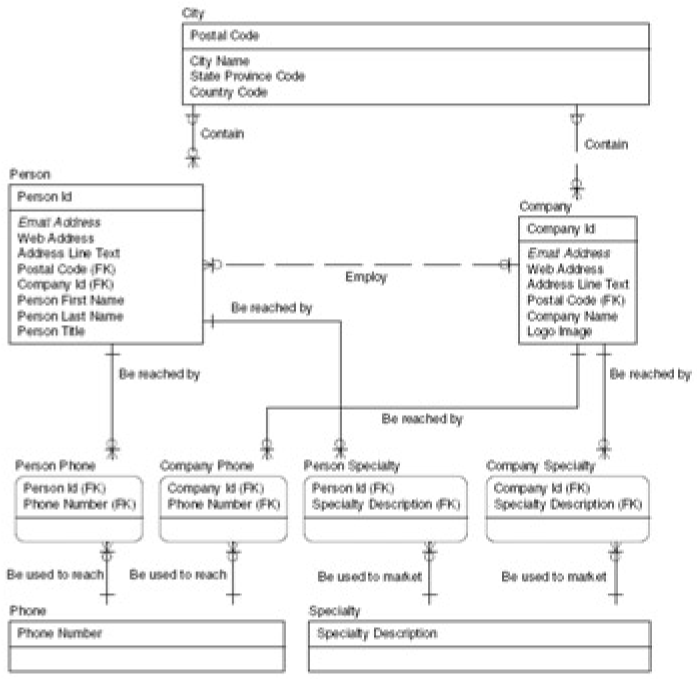


Figure 10.3: Rolling down method of resolving subtyping

In the rolling down technique, we no longer have the Contact concept and everything at the supertype level was copied down into each subtype. Notice that although we have the user friendly concepts of Person and Company, we also have the redundancy of extra relationships to Phone, Specialty, and City, as well as repeating **Email Address, Web Address**, and address information for each subtype.

Rolling up is when each subtype is 'rolled up' into the supertype, moving the data elements and relationships from each subtype up into the supertype. The subtypes disappear and all data elements and relationships only exist at the supertype level. Rolling up adds flexibility to the data model because new types of the supertype can be added, often with no model changes. However, rolling up can also produce a more obscure model, as the audience for the model may not relate to the supertype as well as they would to the subtypes. In addition, we can only enforce business rules at the supertype level, not the subtype level. See Figure 10.4 for rolling up in our contact example.

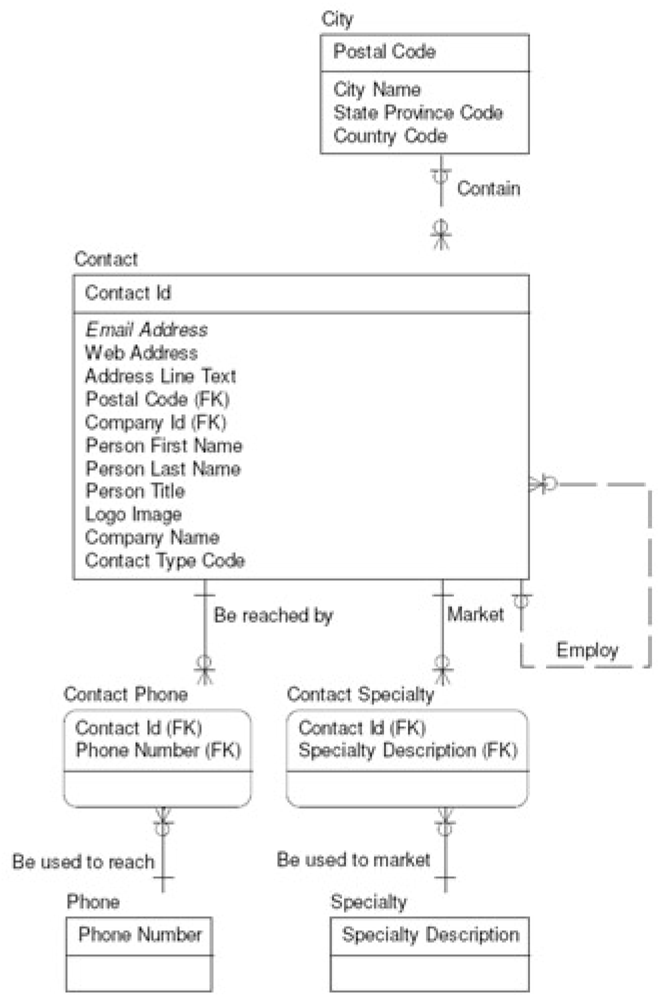


Figure 10.4: Rolling up method of resolving subtyping

When we roll up, we need a way to distinguish the original subtypes from each other. Therefore, we add a data element that distinguishes people from companies, in this case **Contact Type Code**. Two of the values of **Contact Type Code** would represent 'Person' and 'Company'.

Using rolling up, we still retain the business rules for Contact, yet lose the rules that were only enforced for Person or Company. For example, we cannot make **Company Name** a mandatory data element in Contact because a Person does not have a **Company Name** and may not be assigned to a Company.

In addition to choosing denormalization because of the need for faster retrieval time or for more user friendly structures, the standard way of denormalizing can be chosen in the following situations:

- **When you need to maintain the flexibility of the normalized model.** Folding the data elements and relationships together using the standard approach still allows one-to-one and one-to-many relationships to exist. In Figure 10.4 for example, we did not lose the flexibility that a Contact can be a Person or Company (it is just harder to see and enforce).

- **When you want to reduce development time and complexity.** Often there is a direct relationship to the number of tables and relationships on a model, and the amount of effort to develop the application. A developer will need to write code that jumps from one table to another to collect certain data elements, and this can take time and add complexity. Denormalizing into fewer tables using the standard method means the data elements and relationships from different entities now exist in the same entity. In Figure 10.4 for example, if the developer needs to retrieve both the person and company name, they can easily do so from the same entity Contact.

FUBES

FUBES (fold up but easily separate) is an acronym I made up for a technique that uses the standard method of denormalizing while also allowing access to just the data elements from the parent side of a one-to-many relationship. There is an additional data element that contains a level code and additional instances for each of the parents. In Figure 10.5 for example, we have a subset of the logical data model for Calendar.

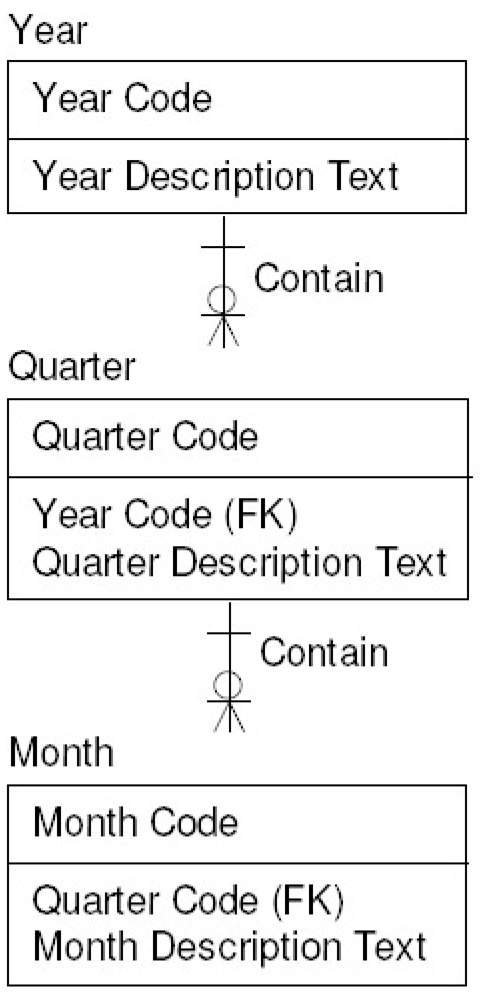


Figure 10.5: Subset of the Calendar logical data model

Table 10.2 contains sample values for each of these entities.

Table 10.2: Sample values for Figure 10.5

➡Open table as spreadsheet

Year	
Year Code	**Year Description Text**
2007	Two Thousand Seven
2008	Two Thousand Eight
2009	Two Thousand Nine

➡Open table as spreadsheet

Quarter		

Quarter Code	Year Code	Quarter Description Text
Q12009	2009	First Quarter Two Thousand Nine
Q22009	2009	Second Quarter Two Thousand Nine
Q32009	2009	Third Quarter Two Thousand Nine

➡Open table as spreadsheet

Month		
Month Code	Quarter Code	Month Description Text
Jan2009	Q12009	January Two Thousand Nine
Feb2009	Q12009	February Two Thousand Nine
Mar2009	Q12009	March Two Thousand Nine

If we decide to denormalize these three entities using the FUBES option, we would have the one entity in Figure 10.6.

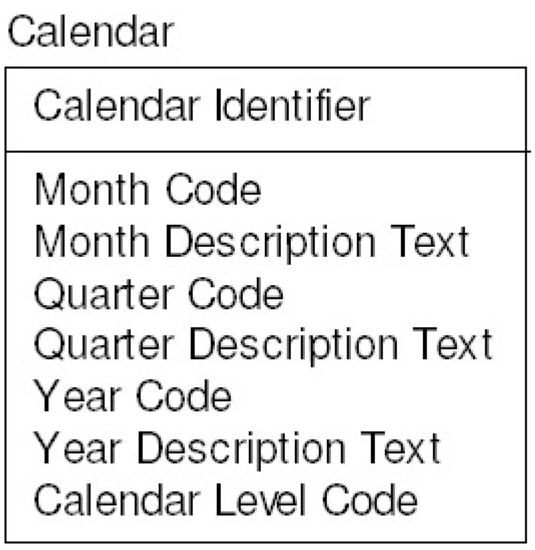


Figure 10.6: FUBES being applied to Calendar

Table 10.3 contains sample values for this table.

Table 10.3: Sample values for Figure 10.6

➡Open table as spreadsheet

Cal Id	Mo Code	Mo Desc	Qtr Code	Qtr Desc	Yr Cd	Yr Desc	Level Cd
1					2007	Two Thousand	Year

Table 10.3: Sample values for Figure 10.6

➡Open table as spreadsheet

Cal Id	Mo Code	Mo Desc	Qtr Code	Qtr Desc	Yr Cd	Yr Desc	Level Cd
						Seven	
2					2008	Two Thousand Eight	Year
3					2009	Two Thousand Nine	Year
4			Q12009	First Quarter Two Thousand Nine	2009	Two Thousand Nine	Quarter
5			Q22009	Second Quarter Two Thousand Nine	2009	Two Thousand Nine	Quarter
6			Q32009	Third Quarter Two Thousand Nine	2009	Two Thousand Nine	Quarter
7	Jan2009	January Two Thousand Nine	Q12009	First Quarter Two Thousand Nine	2009	Two Thousand Nine	Month
8	Feb2009	February Two Thousand Nine	Q12009	First Quarter Two Thousand Nine	2009	Two Thousand Nine	Month
9	Mar2009	March Two Thousand Nine	Q12009	First Quarter Two Thousand Nine	2009	Two Thousand Nine	Month

You may be wondering what value FUBES provides. After all, we are adding a substantial amount of redundancy when using FUBES, because we have all of the redundancy of the standard option plus we are repeating each parent for the child, and repeating all of the parents for that parent. That is, repeating each quarter in our example, plus repeating year for each quarter.

FUBES should be chosen when there is value in denormalizing, but there is a still a need to access parent instances. Having an instance for each parent allows us to achieve better report performance, as we can directly tie to parent levels without having to roll up from the child. We can store sales at a year level, for example, and save the time of summarizing monthly level sales up into a year level. The value is that results can be retrieved extremely fast if we are doing reporting at the parent levels, in this example at the quarter and year level. For example, see Figure 10.7.

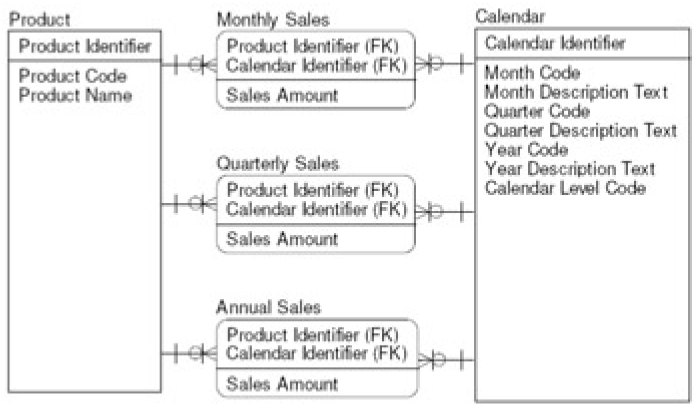


Figure 10.7: Sales reporting with the Calendar FUBES structure

In Figure 10.7 we are summing and loading sales data at the month, quarter, and year levels. If there is a need to retrieve Annual sales for 2008, we can quickly find this number because we are storing the **Sales Amount** for 2008 in Annual Sales, which references the Calendar instance where **Calendar Level Code** equals 'Year' and Year Code equals '2008'.

Repeating Groups

In the repeating-groups technique, the same data element or group of data elements is repeated two or more times in the same entity. Also known as an *array*, a repeating group requires making the number of times something can occur static. Recall that in 1NF we removed repeating groups. Well, here it is, being reintroduced again. An example of a repeating group appears in Figure 10.8.

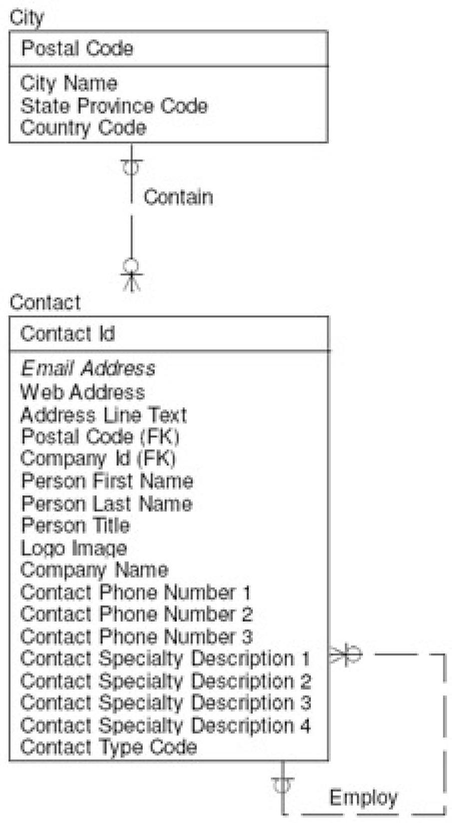


Figure 10.8: Repeating groups in our Contact example

In this example, we can have up to three phone numbers for a Contact and up to four specialties. In addition to choosing denormalization because of the need for faster retrieval time or for more user friendly structures, repeating groups may be chosen in the following situations:

- **When it makes more sense to keep the parent entity instead of the child entity.** When the parent entity is going to be used more frequently than the child entity, or if there are rules and data elements to preserve in the parent entity format, it makes more sense to keep the parent entity. For example, in Figure 10.8, if Contact is going to be accessed frequently, and when Contact is accessed, **Phone Number** and **Specialty Description** are also occasionally accessed.
- **When an entity instance will never exceed the fixed number of data elements added.** In Figure 10.8, we are only allowing up to three phone numbers for a contact, for example. If we have four phone numbers for Bob the Contact, how would we handle this? Right now, we would have to pick which three of them to store.
- **When you need a spreadsheet.** A common use is to represent a report that needs to be displayed in a spreadsheet format. For example, if a user is expecting to see a sales report in which sales is reported by month for the last 12 months, an example of a repeating group containing this information is shown in Figure 10.9. At the end of a given month, the oldest value is removed and a new value is added. This is called a *rolling 12 months*. Faster performance and a more user-friendly structure lead us to add repeating groups in this example and purposely violate 1NF.

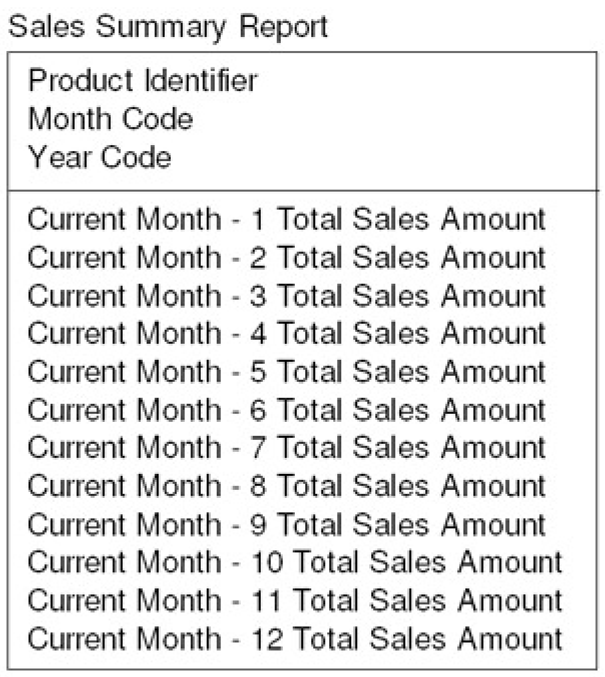

Sales Summary Report
Product Identifier
Month Code
Year Code
Current Month - 1 Total Sales Amount
Current Month - 2 Total Sales Amount
Current Month - 3 Total Sales Amount
Current Month - 4 Total Sales Amount
Current Month - 5 Total Sales Amount
Current Month - 6 Total Sales Amount
Current Month - 7 Total Sales Amount
Current Month - 8 Total Sales Amount
Current Month - 9 Total Sales Amount
Current Month - 10 Total Sales Amount
Current Month - 11 Total Sales Amount
Current Month - 12 Total Sales Amount

Figure 10.9: Sales-report entity with repeating group

Repeating Data Elements

Repeating data elements is a technique in which you copy one or more data elements from one entity into one or more other entities. It is done primarily for performance because by repeating data elements across entities, we can reduce the amount of time it takes to return results. If we copy **Customer Last Name** from the Customer entity to the Order entity, for example, we avoid navigating back to Customer whenever just **Customer Last Name** is needed for display with order information.

Repeating data elements differs from repeating groups because the repeating groups technique replaces one or more entities and relationships, while the repeating data elements technique retains the existing entities and relationships and just copies over the data elements that are needed. For example, to apply the repeating groups technique to Customer and Order, we'd need to determine the maximum number of orders a customer can place, then remove the Order entity and its relationship to Customer and repeat three or four or however many times all of the order data elements within Customer. The repeating data elements technique would just involve copying over the data elements we need and keeping everything else intact.

See Figure 10.10 for an example of this technique using our contact example. In this example, there was a need to view the country along with contact information. Therefore only the **Country Code** data element needed to be copied to Contact and the City entity and its relationship remains intact.

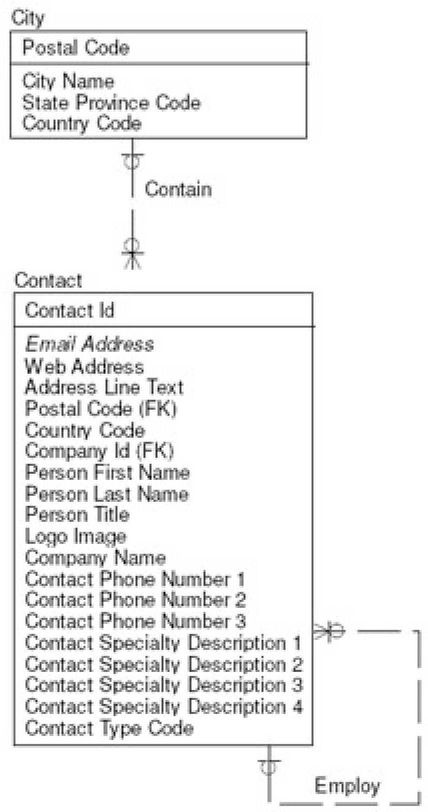


Figure 10.10: Repeating data elements in our Contact example

In addition to choosing denormalization because of the need for faster retrieval time or for more user friendly structures, repeating data elements can be chosen in the following situations:

- **When the repeated data element or elements are accessed often and experience little or no changes over time.** If for example, **Country Code** changed frequently, we would be required to update this value on City and for each of the contacts related to City. This takes time to update and could introduce data quality issues if not all updates were performed correctly or completely.
- **When the standard denormalization option is preferred but space is an issue.** There might be too huge an impact on storage space if the entire parent entity was folded up into the child and repeated for each child value. Therefore, only repeat those parent data elements that will be used frequently with the child.
- **When there is a need to enforce the business rules from the LDM.** With the repeating data elements technique, the relationships still remain intact. Therefore, the business rules from the logical data model can be enforced on the physical data model. In Figure 10.10 for example, we can still enforce that a Contact live in a valid City.

Summarization

Summarization is when tables are created with less detail than what is available in the business. Monthly Sales, Quarterly Sales, and Annual Sales from Figure 10.7 are all summary tables derived from the actual order transactions.

In addition to choosing denormalization because of the need for faster retrieval time or for more user friendly structures, summarization can be chosen when there is a need to report on higher levels of granularity than what is captured on the logical data model. Annual Sales for example, provides high level summary data for the user, and therefore the user (or reporting tool) does not have to spend time figuring out how to produce annual sales from detailed tables. The response time is much quicker because time does not need to be spent summarizing data when it is requested by a user; it is already at the needed summarization level, ready to be queried.

Star Schema

Denormalization is a term that is applied exclusively to relational physical models, because you can't denormalize something unless it has already been normalized. However, denormalization techniques can be applied to dimensional models, as well - you just can't use the term 'denormalization'. So all of the techniques from the standard method through summarization can be used in dimensional modeling, just choose a term such as 'flattening' instead of 'denormalization'.

A star schema is the most common dimensional physical data model structure. The term 'meter' from the dimensional logical data model is replaced with the term 'fact table' on the dimensional physical data model. A star schema results when each set of tables that make up a dimension is flattened into a single table. The fact table is in the center of the model and each of the dimensions relate to the fact table at the lowest level of detail. A star schema is relatively easy to create and implement, and visually appears elegant and simplistic to both IT and the business.

Recall the dimensional logical modeling example from the previous chapter, copied here as Figure 10.11.

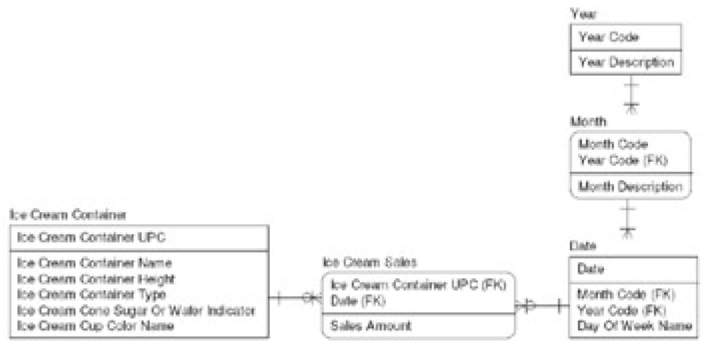


Figure 10.11: Dimensional logical data model of ice cream

A star schema of this model would involve folding Year into Month and Month into Date, and then renaming Date with a concept inclusive of Month and Year, such as 'Time' or 'Calendar'. See Figure 10.12 for the star schema for this example.

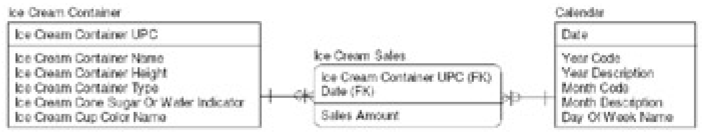


Figure 10.12: Ice cream star schema

Views Explained

A view is a virtual table. It is a dynamic "view" or window into one or more tables (or other views) where the actual data is stored. A view is defined by a query that specifies how to collate data from its underlying tables to form an object that looks and acts like a table but doesn't physically contain data. A query is a request that a user (or reporting tool) makes of the database, such as "Bring me back all **Customer Ids** where the Customer is 90 days or more behind in their bill payments." The difference between a query and a view, however, is that the instructions in a view are already prepared for the user (or reporting tool) and stored in the database as the definition of the view, whereas a query is not stored in the database and may need to be written each time a question is asked.

Returning to our contact example, let's assume the users continuously ask the question "Who are the people that live in New Jersey?" We can answer this question in a view. Figure 10.13 contains the model from Figure 10.10 with the addition of a view to answer this question.

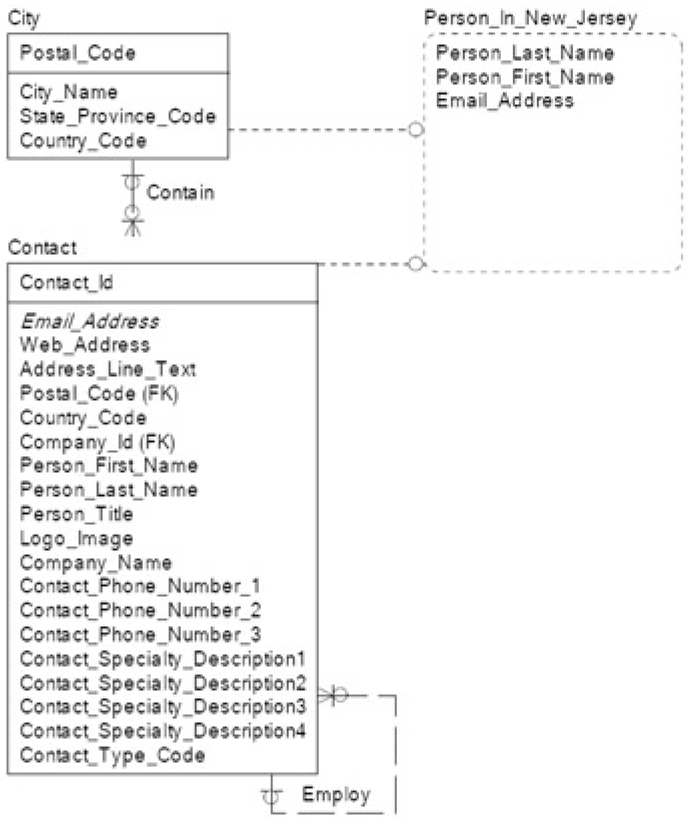


Figure 10.13: View added to answer business question

In many tools, a view is shown as a dotted box. In this model, it is called Person_In_New_Jersey. This view needs to bring together information from the entity City and from the entity Contact to answer the business question. The instructions to answer this business question are written in a query language a database can understand, usually in the language SQL (pronounced 'sequel'). Figure 10.14 contains the SQL language to answer this business question.

```
SELECT Contact.Person_Last_Name,
Contact.Person_First_Name, Contact.Email_Address

FROM City, Contact

WHERE (City.Postal_Code=Contact.Postal_Code)

AND (City.State_Province_Code='NJ')

AND (Contact.Contact_Type_Code = 'Prsn');
```

Figure 10.14: SQL language to answer question "Who are my Contacts that live in New Jersey?"

SQL is powerful for the same reason that data modeling is powerful: with a handful of symbols, one can communicate a lot. In English, this SQL statement is saying "Give me the last name, first name and email address of the Contact(s) whose postal code matches a postal code in the City table that has a state code of 'NJ' and who is a person (and not a company)".

There are different types of views. Typically, execution of the view (i.e. the SQL statement) to retrieve data takes place only when a data element in the view is requested. It can take quite a bit of time to retrieve data, depending on the complexity of the request and the data volume. However, there are types of views, such as materialized views or snapshots, that can match and sometimes even beat retrieval speed from actual tables, because their instructions are run at a predetermined time with the results stored in the database, similar to a database table.

Views are a popular choice for assisting with security and user-friendliness. If there are sensitive data elements within a database table that only certain people in the company have access to, then views are a great way to hide these sensitive data elements from the common user. Views can also take some of the complexities out of joining tables for the user or reporting tool.

In fact, we can use views in almost all situations where we are using denormalization. At times, views can offer all of the benefits of denormalization *without the drawbacks associated with data redundancy and loss of referential integrity*. A view can provide user-friendly structures over a normalized structure, thereby preserving flexibility and referential integrity. A view will keep the underlying normalized model intact, and at the same time present a denormalized or flattened view of the world to the business.

Indexing Explained

An index is a pointer to something that needs to be retrieved. An analogy often used is the card catalog, which in the library, points you to the book you need. The card catalog will point you to the place where the actual book is on the shelf, a process that is much quicker than looking through each book in the library until you find the one you need. Indexing works the same way with data. The index points directly to the place on the disk where the data is stored, thus reducing retrieval time. Indexes work best on data elements whose values are requested frequently but rarely updated.

Primary keys, foreign keys, and alternate keys are automatically indexed just because they are keys. A non-unique index, also known as a secondary key, is when an index based on one or more non-key data elements is added to improve retrieval performance. Where to add non-unique indexes depends on the types of queries being performed against the table. For example, recall Figure 10.12 repeated here as Figure 10.15.

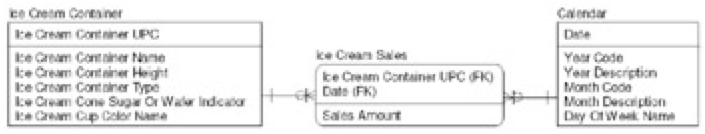


Figure 10.15: Ice cream star schema

Assume that an often-asked question of this dimensional model is "What are my sales by Container Type?" Or in more business speak, "What are my sales by Ice Cream Cup versus Ice Cream Cone?" Because this query involves the data element Ice Cream Container Type, and because Ice Cream Container Type is most likely a stable data element which does not experience many value updates, Ice Cream Container Type would be a good candidate for a secondary index.

Partitioning Explained

In general, a partition is a structure that divides or separates. Specific to the physical design, partitioning is used to break a table into rows, columns or both. An attribute or value of an attribute drives how the records in a table are divided among the partitions. There are two types of partitioning - vertical and horizontal. To understand the difference between these two types, visualize a physical entity in a spreadsheet format where the data elements are the columns in the spreadsheet and the entity instances are the rows. Vertical means up and down. So vertical partitioning means separating the columns (the data elements) into separate tables. Horizontal means side to side. So horizontal partitioning means separating rows (the entity instances) into separate tables.

An example of horizontal partitioning appears in Figure 10.16. This is our data model from Figure 10.10, modified for horizontal partitioning. In this example, we are horizontally partitioning by **Contact Last Name**. If a contact's last name starts with a letter from 'A' up to and including 'H', then the contact would appear as an instance of the entity Contact A Through H. If a contact's last name starts with a letter from 'I' up to and including 'Z', then the contact would appear as an instance of the entity Contact I Through Z.

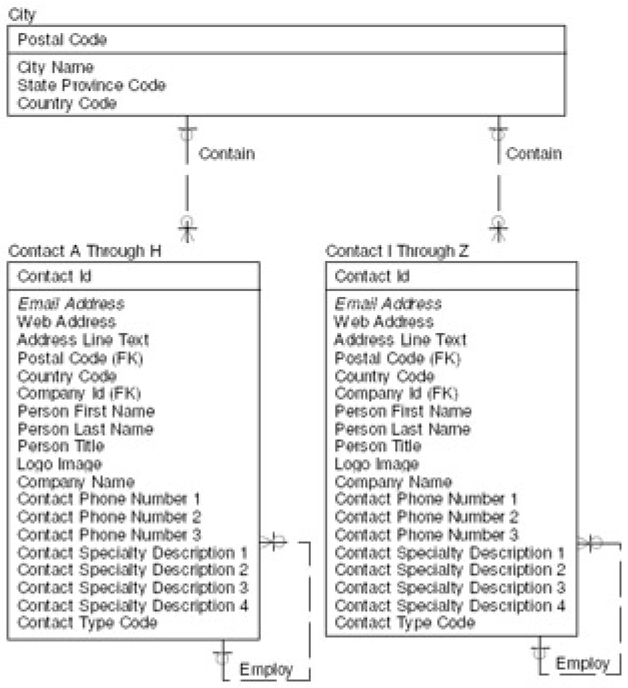


Figure 10.16: Horizontal partitioning in our contact example

An example of vertical partitioning appears in Figure 10.17. Also, based on the example from Figure 10.10, we have vertically partitioned the phone numbers into a separate table and the specialties into a separate table. This might have been done for space or user access reasons.

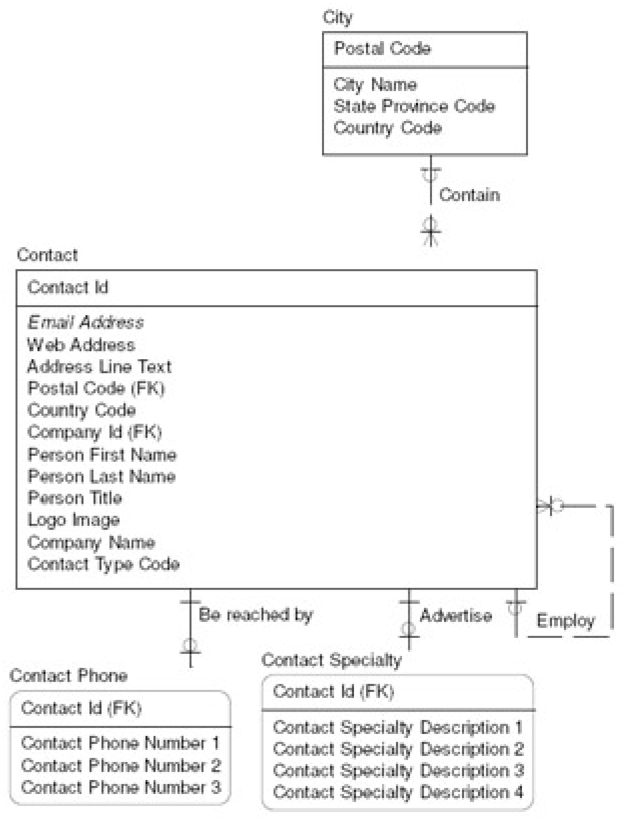


Figure 10.17: Vertical partitioning in our contact example

Snowflake

In dimensional modeling, there are two main types of physical designs: the star schema and snowflake. We mentioned earlier that the star schema is when each dimension has been flattened into a single table. The snowflake is when there are one or more tables for each dimension. Sometimes the snowflake structure is equivalent to the dimensional logical model, where each level in a dimension hierarchy exists as its own table. Sometimes in a snowflake, however, there can be even more tables than exist on the dimensional logical model. This is because vertical partitioning is applied to the dimensional model.

For example, Figure 10.18 is based on our ice cream dimensional model from Figure 10.11. This is a snowflake. Not only does the calendar dimension exist in separate tables (one for year, one for month, one for date), but we have vertically partitioned the **Ice Cream Cone Sugar Or Wafer Indicator** into the Ice Cream Cone entity, and vertically partitioned the **Ice Cream Cup Color Name** into the Ice Cream Cup entity. Notice that in this example, vertical partitioning is equivalent to the Identity method of resolving a subtype. Another way of saying this is that Identity is a type of vertical partitioning.

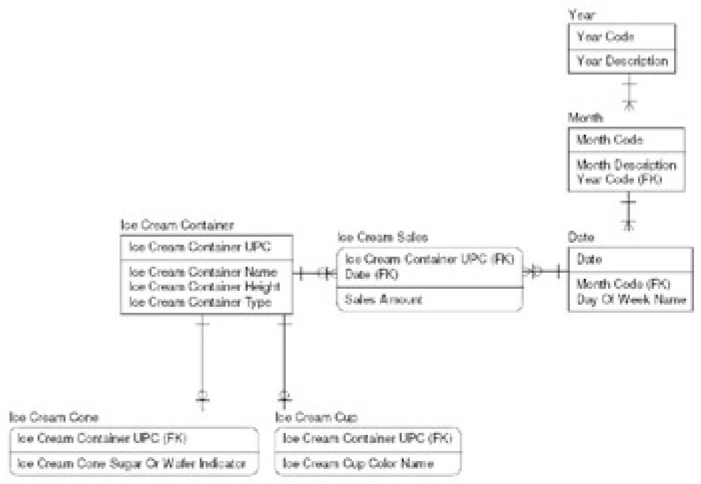


Figure 10.18: Ice cream snowflake

Almost all dimensional logical data models become star schemas in the physical world. Therefore, it is rare to see a snowflake design. However, it does occur from time to time for the two main reasons of data volatility and the need for higher level access.

Data volatility means that values are updated frequently. If we have a four level product hierarchy on a dimensional model, for example, and all four levels are relatively stable except for the second level, which experiences almost daily changes, vertically partitioning to keep this second level separate can reduce database complexity and improve database efficiencies.

In dimensional modeling, users often want to query across data marts, so dimensions need to be built consistently across them. One data mart might need to see facts at different grains than another data mart. For example, one data mart might need to see **Gross Sales Amount** at a date level, and another data mart might only require seeing **Gross Sales Amount** at a year level. Keeping the calendar structure as on the dimensional logical model with separate tables for Date, Month, and Year would allow each data mart to connect to the level they need. Another option is to use the FUBES techniques discussed previously. This would allow the modeler to use a star schema design, storing all calendar data elements in one table and accessing different levels using the **Calendar Level Code** data element.

When Reference Data Values Change

Transaction data elements are those that capture data on the events that take place in our organizations and reference data elements are those that capture the context around these events. For example, an order is placed for '5'. The order itself is a transaction and data elements such as **Order Number** and **Order Quantity** are transaction data elements. However '5', does not provide any value unless we give it context. Product information, Customer information, Calendar information, etc., provides the context for '5' — all of these data elements are reference data. *Bob ordered 5 Widgets on April 15th, 2009*. 'Bob' is the customer reference data, 'Widgets' is the product reference data, and 'April 15th, 2009' is the calendar reference data.

Transaction data and reference data behave very differently from each other in terms of value updates. Transaction data occurs very frequently and usually once they occur they are not updated (e.g. once an order has been delivered you can no longer change its properties such as **Order Quantity**). Reference data, on the other hand, is much less voluminous but values can change more often (e.g. people move to different addresses, product names change, and sales departments have reorganizations).

Reference entity instances will therefore experience changes over time, such as a person moving to a new address, or product name changing, or an account description being updated. There are three ways for modeling changes to data values:

- **Store only the most current information.** When values are updated, store only the new values. For example, if Bob the customer moves to a new address, store just his new address, and do not store any previous addresses.
- **Store the most current along with all history.** When values are updated, store the new values along with any previous values. If Bob the customer moves to a new address, store his new address along with all previous addresses. New entity instances are created when values are updated and previous entity instances remain intact.
- **Store the most current and some history.** When values are updated, store the new values along with some of the previous values. If Bob the customer moves to a new address, store his new address along with only his previous address. New data elements are added to store the changes.

These three ways of handling changing values can be applied to both relational and dimensional models. On dimensional models, there is a special term that describes how to handle changing values: Slowly Changing Dimension (SCD). An SCD of Type 1 means only the most current information will be stored. An SCD of Type 2 means the most current along with all history. And an SCD of Type 3 means the most current and some history will be stored.

Using the Contact entity from Figure 10.18 as an example, Figure 10.19 shows all three types of SCDs.

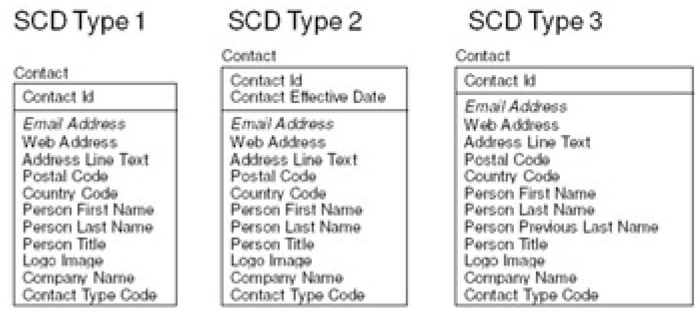


Figure 10.19: The three types of SCDs

The SCD Type 1 is just a current view of Contact. If there are any updates to a Contact, only the updates will be reflected and not the original information. The SCD Type 2 contains the most current data as well as a full historical view of Contact. If a Contact instance experiences a change, the original contact instance remains intact and a new instance is created with the most current information. The SCD Type 3 includes only a current view of Contact with the exception of the person's last name where we have a requirement to also see the person's previous last name. So if Bob changes his last name five times, the SCD Type 1 will just store his current last name, the SCD Type 2 will store all five last names, and the SCD Type 3 will store the current last name, along with the most recent last name.

Exercise 10: Getting Physical with Subtypes

1. Subtyping is a powerful communication tool on the logical data model, because it allows the modeler to represent the similarities that exist between distinct business concepts to improve integration and data quality. As an example, refer to Figure 10.20 where the supertype Course contains the common data elements and relationships for the Lecture and Workshop subtypes.

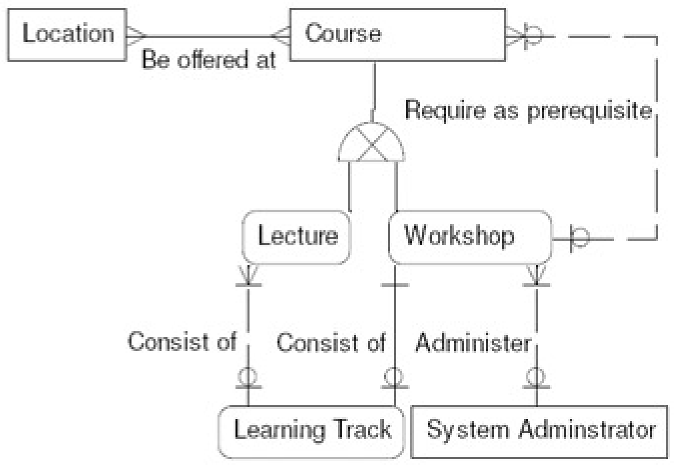


Figure 10.20: Course subtyping structure

Briefly walking through this model, we learn that each Course, such as Data Modeling 101, can be either a Lecture or a Workshop. Each Course must be taught at many Locations. Each Learning Track, such as the Data Track, must consist of one or many Lectures, yet only one Workshop. Each Workshop, such as the Advanced Design Workshop, can require certain Courses as a prerequisite, such as Data Modeling 101 and Data Modeling 101 Workshop. Each System Administrator must administer one or many Workshops.

Assume this is a logical data model (with data elements hidden to keep this example manageable). On the physical data model, we can replace this subtype symbol with one of three options:

- **Rolling down.** Remove the supertype entity and copy all of the data elements and relationships from the supertype to each of the subtypes.
- **Rolling up.** Remove the subtypes and copy all of the data elements and relationships from each subtype to the supertype. Also add a type code to distinguish the subtypes.

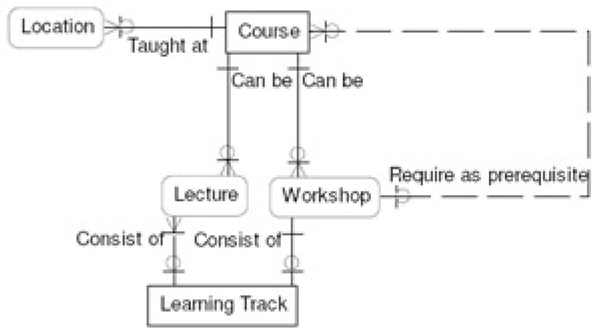
- **Identity.** Convert the subtype symbol into a series of one-to-one relationships, connecting the supertype to each of the subtypes.

For this Challenge, using the Course structure from Figure 10.20, build all three options.

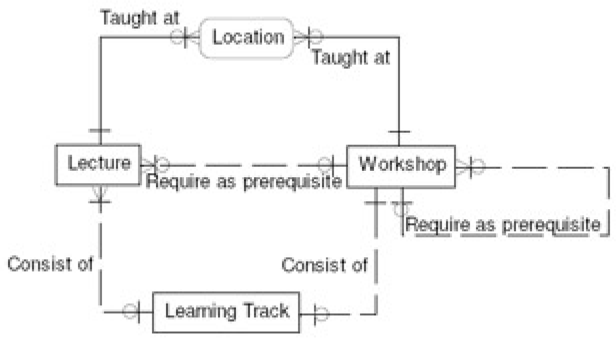
Answers

1. The following are the three different ways this subtyping structure can be represented on the physical data model.

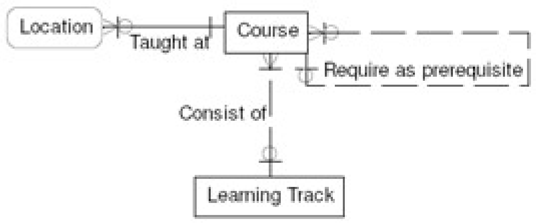
IDENTITY



ROLLING DOWN



ROLLING UP



See the Appendix for my answers.

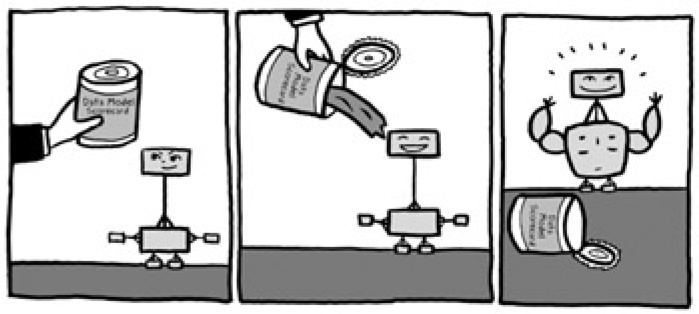
Key Points

- The physical data model builds upon the logical data model to produce a technical solution.
- Denormalization is the process of selectively violating normalization rules and reintroducing redundancy into the model.
- There are five denormalization techniques: standard, repeating groups, repeating data elements, FUBES, and summarization.
- A star schema is when each set of tables that make up a dimension is flattened into a single table.
- A view is a virtual table.
- An index is a pointer to something that needs to be retrieved.
- Partitioning is breaking up a table into rows, columns or both. If a table is broken up into columns, the partitioning is vertical. If a table is broken into rows, the partitioning is horizontal.
- Snowflaking is when vertical partitioning is performed on a dimensional model, often due to data volatility or frequent user access to higher levels in a hierarchy.
- Data values change over time. We have the option of storing only the new information (Type I), storing all of the new information plus all of the historical information (Type II), or storing the new information plus some of the historical information (Type III).

Section IV: Data Model Quality

Chapter List

Part Overview



By now, you know the important role data modeling plays in application development, you know the components of a data model, and you know the six different types of data models. Now that we have all this data modeling knowledge, how can we improve the quality of our data modeling deliverables? Section IV focuses on improving data model quality through templates, the Data Model Scorecard®, and better communication with businesspeople and the project team.

Chapter 11 goes through some useful templates for capturing and validating requirements. These templates will save time as well as improve the accuracy of the resulting data model.

Chapter 12 focuses on the Data Model Scorecard®, a proven technique for validating data model quality. The quality of the data model directly impacts many characteristics of the resulting application, including stability and data quality.

Chapter 13 provides advice for the analyst and data modeler on working with other team members. Graeme Simsion has worked in the data modeling field for twenty-five years as a practitioner, teacher and researcher, and he shares his experiences on setting expectations, staying on track, and achieving closure.

Chapter 11: Which Templates Can Help with Capturing Requirements?

Overview

Recycle, reuse
Easy to fill in templates
Consistency rules

There are a number of templates I use that save modeling time as well as improve the accuracy of the resulting data model. These tools are the focus of this chapter. You don't need to use every tool for every modeling effort—pick and choose the ones that make sense for your environment and the people you're working with.

The In-The-Know Template captures the people and documentation that can contribute and validate the data requirements. The Subject Area List is a compilation of the key concepts the business feels are important to capture, along with their definitions. The Family Tree is a spreadsheet that captures the source applications and other key metadata for each subject area or data element within the scope of our application. The Grain Matrix is a spreadsheet that captures the levels of reporting for each concept or fact.

In-The-Know Template

The In-The-Know Template captures the people and documentation that can provide and validate the data requirements. It is a record of peoples' names and roles and contact information, as well as a list of where other important resources are located. For example, it contains the location of useful documentation for completing data modeling deliverables, such as business and functional requirements. Relying on memory instead of writing the locations down can lead to forgetting how to find the documents. Table 11.1 contains a sample In-The-Know Template.

Table 11.1: Sample In-The-Know Template

➡Open table as spreadsheet

Subject Area	Resource	Type	Role/ How used	Location/Contact
Customer	Tom Jones	Subject Matter Expert	Customer Reference Data Administrator	212-555-1212
	Customer classification list	Reference document	To validate and create new customer classifications	S:/customer/custclsfn.xls
Item	Current Item Report	Report	To identify all current item information	www.item.com

Here is a description on how each of these columns is used:

- **Subject Area.** This is the concept name. You can take the list of concepts from a subject area model or the Subject Area List (the next technique discussed) and put them right into this column. Examples from Table 11.1 are Customer and Item.
- **Resource.** This is the source of the information. In this template, it is broad enough to be anything useful. It includes people, requirements documents, reports, etc. Be as specific as possible in this description column. If there is more than one resource for the same subject area, use a separate row in the template for the additional resources. Examples from Table 11.1 are Tom Jones, Customer classification list, and Current Item Report.
- **Type.** Provides a general categorization of each of the resources. Because this template can be so generic, it is important to put each of the resources into the most appropriate category. Examples from Table 11.1 are Subject Matter Expert, Reference document, and Report. This is an optional category that is most useful when the In-The-Know Template is very large.
- **Role / How used.** This provides why the resource is valuable to this project. Why are we bothering to list this resource in this template? Be specific! Examples from Table 11.1 are Customer Reference Data Administrator, To validate and create new customer classifications, and To identify all current item information.
- **Location / Contact.** This column contains how to reach the resource. If the resource is a document, this column might contain the document's path name on a public drive or where to find it on a server or website. If the resource is a person, this column might contain the person's phone number, email address, or mailing address. Examples from Table 11.1 are 212-555-1212, S:/customer/custclsfn.xls, and www.item.com.

The In-The-Know Template captures the people and documentation that can provide and validate the data requirements. Having this information in a standard format accomplishes several goals:

- **Provides a handy and complete reference list.** This template is easy to read and provides all of the types of information that are necessary to identify people and documentation resources. Even years after the project is in production, this list could still be useful for functional or technical project questions.
- **Finds gaps or redundancy in your available list of resources.** This document will highlight any missing information. For example, if there is no available expert on item information, it will be very evident here. If you are missing a key reference document, you will notice it here and can raise it to management's attention.
- **Acts as a sign off document.** For example, if a person is identified as a resource for a particular subject area, their management can be made aware of this and allocate their time accordingly. This way you will minimize the chances that your customer expert is pulled in another direction when you need their help.

Subject Area List

The Subject Area List is a handy starting point if you feel it might be too much of a jump to start capturing high level business rules—that is, the relationships on a subject area model. The Subject Area List is a compilation of the key concepts the business feels are important to capture, without introducing data modeling notation. As your model develops, you might need to refine terms and definitions on the Subject Area List to keep them up-to-date and valuable. Table 11.2 shows a sample Subject Area List.

Table 11.2: Sample Subject Area List

Open table as spreadsheet

Name	Synonyms	Definition	Questions
Asset	Machine, Part, Capital, Stock, Wealth, Supplies	Something our company owns that is considered to be valuable.	Does this also include purchases we plan on making for the upcoming year?
Carrier	Trucking Company, Distributor, Transporter	A company that physically moves our products from one site to another site.	Does carrier include both company-owned carriers and externally-owned? Or just our own carriers?
Company	Corporation, Firm, Organization, Partnership	A business enterprise that serves a purpose to society and the economy.	Are our business units or subsidiaries considered separate companies within a larger company, or do we just have one company?
Contract	Order, Promotion, Agreement, Invoice, Bill of	A document containing the terms and	What is the difference between a transaction and a contract?

Table 11.2: Sample Subject Area List

➡Open table as spreadsheet

Name	Synonyms	Definition	Questions
	Lading, Bill of Materials, Purchase Order, Policy, Statement	conditions around the purchase or creation of product.	Which happens first?

Here is a description of each of the columns:

- **Name.** Name contains the most common term for each subject area. Ideally, this will be an enterprise-wide agreed-upon name for this subject area. Subject areas are listed alphabetically by name.
- **Synonyms.** Synonyms are a great place to list aliases for this term. When there is more than one name for the same subject area, listing all of the names under Synonyms is a step towards reaching a single name and definition. This column also includes those words that are more specific to a particular industry, or those terms that are at a more detailed subject area level than the subject area term. For example, Order in this column is a more specific term for the Contract subject area.
- **Definition.** Definition contains a brief description of each of the subject areas. The basic definition is designed to be generic enough to be easily customizable, yet detailed enough to provide value.
- **Questions.** Questions contain some queries or comments that might be worthwhile addressing when refining your subject area definitions. This last column can spark lively discussion and debate, resulting in a more solid subject area definition. Your goal for these questions is to more clearly articulate the definition of a subject area by answering what that subject area does and does not include.

The Subject Area List is a very simple tool which accomplishes a number of important goals:

- **Creates a high-level understanding of the application's subject areas.** After completing this checklist, you, and the rest of the project team, will clearly understand each subject area and know what is within the scope of the application. When you start the modeling process, you will already know the subject areas and only have to add the relationships between these subject areas to complete the model.
- **Gets your project team 'out of the weeds'.** This is useful when your project team wants to start off the modeling phase by diving directly into the data elements. In these situations, it can be very easy to miss key subject areas and accidentally leave out required information. Starting the application analysis by listing data elements creates a narrow and incomplete view of the project's requirements. In order to step back and really understand the scope of the application, your team can appreciate the broad and complete picture through a subject area view.
- **Facilitates entity and data element names and definitions.** Having solid names and definitions at the subject area level makes it easier to develop names and definitions at more detailed levels. If we have a standard name and very good definition for Customer,

for example, we can name and define the entities Customer Location and Customer Association, and the data elements **Customer Last Name** and **Customer Type Code** more easily.

- **Initiates rapport with the project team and business users.** Completing the Subject Area List as a group is a very good first data modeling task for you and the rest of the team. It is a quick win, meaning it is not usually very difficult to complete, yet has big payoffs. It can improve everyone's understanding of the new application's content at a high-level, as well as help build rapport with the project team.

Family Tree

The Family Tree is a spreadsheet that captures source information and other key descriptive information for each subject area or data element within the scope of our application. The greater the number of source systems, the more valuable the Family Tree. The Family Tree works especially well when our application is a data mart, because this tool can capture which information currently exists in the data warehouse and which applications we will require new information from.

Keeping the sourcing information neatly organized within a spreadsheet makes validation much easier than sifting through pages in a requirements document. A spreadsheet is an extremely unambiguous way to represent the sourcing requirements. Once the subject area or data element source is validated and agreed upon, the Family Tree is still useful as a reference. Long after the application is developed and in production, people will still need to refer to this information. Table 11.3 contains an example of a Family Tree at a subject area level.

Table 11.3: Sample Subject Area Family Tree

Open table as spreadsheet

From Here				**To Arrive Here**		
Name	**Source**	**Definition**	**History**	**Name**	**Definition**	**History**
Customer	Customer Reference Database	The recipient and purchaser of our products.	10	Customer	The recipient and purchaser of our products.	3
Order	Order Entry	A contract to buy a quantity of product at a specified price.	4	Order	A contract to buy a quantity of product at a specified price.	3
Item	Item Reference Database	Anything we buy, sell, stock, move, or make. Any manufactured or purchased part, material, component, assembly, or product.	10	Product	Anything we sell.	3
Party	Data Warehouse	A person or company that is important to our	5	Associate	A person who is employed in a full	3

Table 11.3: Sample Subject Area Family Tree

➡Open table as spreadsheet

From Here				To Arrive Here		
Name	**Source**	**Definition**	**History**	**Name**	**Definition**	**History**
		business in some way.			or part time capacity by our company, to perform a service or function to our company.	
Time	Data Warehouse	A measurable length of seconds, minutes, hours, days, weeks, months or years. Can include both fiscal and Julian time measurements.	10	Calendar	A measurable length of days.	3

Here is a description of each of the columns in this spreadsheet:

- **Name.** This column contains the names of all the subject areas or data elements within the application. If a subject area or data element is left out of this document, there is a good chance that it will not be in the data model. Name needs to be specified on both the "From Here" and "To Arrive Here" sections of the spreadsheet because there may be different names for each. For example, Item on the "From Here" side and Product on the "To Arrive Here" side on Table 11.3.
- **Source.** This column contains the name of the application that provides the data for each subject area or data element. Note that a source application is not just limited to a database, but can also be a file or a spreadsheet. A very important question to answer at this point is "How far should I go back?" That is, should we list all of the source applications until we reach the point where a user is typing the information in? Or should we just go back to the immediate source? My advice is to continue following the subject area or data element upstream until you arrive at a reliable and accurate source for this subject area or data element, and then stop. On Table 11.3 for example, in order to get Associate information, we need to go back to the data warehouse, which stores Associate as Party.
- **Definition.** Just as we saw with the Name column, on this tree, we can have multiple definitions of a single subject area or data element. However, there is also a slim chance that one of the source systems might have a definition slightly different from the enterprise definition. On Table 11.3 for example, the definition of Item on the "From Here" side of the spreadsheet is much broader than the definition of Product on the "To Arrive Here" side of the spreadsheet.
- **History.** The History column under the source set of columns is the number of years of history the system maintains on this subject area or data element. Luckily, in our example

on Table 11.3, the history needs can all be met. The number of years required on the "To Arrive Here" side is always less than the number of years available on the "From Here" side. If we want more history than is available in our source system, we have an issue. Working with our users, we may be able to agree that in the beginning, there will be less history, and over time, the history will accumulate to the level they are requiring. Finding this history gap early catches a potentially serious problem before much time has been invested in the project and helps set the user's expectations. This column is optional for operational applications, but mandatory for business intelligence applications.

Additional types of descriptive information may be included in The Family Tree, especially at a data element level. For example, it can contain formatting information such as data element length and domain.

The Family Tree has several objectives:

- **Captures each subject area's or data element's source information.** The Family Tree contains each of the applications that play a role in transporting and shaping the content of each of the subject areas or data element.
- **Estimates work effort.** Before starting development, you can gauge pretty accurately how much effort it would take to get the information required. You can judge how many applications are involved and the impact each application will have on your development and vice versa. This allows you to create an estimate of the effort it will take to bring this new information into the application. Note that if we are developing a data mart, we cannot complete a Grain Matrix before finalizing an estimate. The Grain Matrix will be discussed next.
- **Identifies sourcing issues early.** Let's say your users want three years of history for Customer and by filling in the Family Tree, you realize that the source system only keeps one year of Customer history. This is a problem that you have discovered with a minimal amount of analysis and time. Identifying problems like this one early on is a great benefit of this tool. It takes a proactive approach to sourcing issues.

Grain Matrix

The Grain Matrix is a spreadsheet that captures the levels of reporting needed for each measure. It is the spreadsheet view of a reporting application and acts as a universal translator between business users and the technical project team. Both business users and the project team can understand and validate the reporting requirements through this spreadsheet. Business users might 'speak' the language of reports, whereas technical people might 'speak' the language of data marts. The Grain Matrix is a common ground; a spreadsheet that both users and technical people can relate to and use as a tool for communication.

The Grain Matrix can be used for both ad hoc and predefined reporting. Ad hoc reporting means that the user has the ability to navigate through database tables, often in unpredictable ways, with a substantial amount of freedom over how they manipulate and report on the data. Predefined reporting is where the user is provided with a template or report that is already populated with all of the possible navigation paths and calculations. The Grain Matrix can meet both ad hoc and predefined reporting needs, usually with different labels on the same matrix.

As with the Family Tree, the Grain Matrix can be defined at the subject area or data element level. Table 11.4 contains a Subject Area Grain Matrix after having been filled in initially. Note the different notation used for ad hoc versus the predefined production and shipments summary reports.

Table 11.4: Sample Subject Area Grain Matrix

➡Open table as spreadsheet

Subject Area	**Calendar**				**Customer**		**Product**	
	Year	Quarter	Month	Date	Category	Customer	Brand	Item
Shipments	AB	AB	AB	A	AB	A	AB	A
Debits	A	A	A	A	A	A		
Credits	A	A	A	A	A	A		
Balances	A	A	A	A			A	A
Production	AC	AC	A	A			AC	AC
Contact	A	A	A	A	A	A	A	A

LEGEND:

A = Ad hoc reporting, B = Shipments Summary Report, C = Production Summary Report

So in this example, for ad hoc reporting, there is a need to view all subject areas down to a Date level, but only Shipments, Debits, Credits, and Contact down to a Customer level, and Shipments, Balances, Production, and Contact down to an Item level. The Shipments Summary Report requires Shipments to be at Month, Category, and Brand levels. The Production Summary Report requires Production down to Quarter and Item levels.

The Grain Matrix acts as a universal translator for data mart reporting requirements. The common language created on this spreadsheet by business users and the technical team accomplishes several goals:

- **Facilitates discussion and agreement on the levels of reporting without showing a data model.** Many users I've worked with have difficulty understanding data modeling concepts. This causes them confusion and frustration when using a dimensional data model to validate reporting requirements. Rather than use the data model as a validation mechanism, I prefer to use the Grain Matrix. Not too long ago, I validated all of the reporting requirements for a financial data mart with a business user, without showing a single data model. I've found all users are very comfortable with a spreadsheet. The spreadsheet format, as we saw with the Family Tree, provides an unambiguous format for capturing the reporting requirements. Sometimes looking at report printouts or paragraphs of text in a requirements document can lead to lots of questions or incorrect assumptions. A spreadsheet displays the reporting requirements more clearly, making it easier for agreement and validation.

- **Consolidates reports.** Frequently, after viewing the completed Grain Matrix, we realize that several distinct reports really have a lot in common with each other. We might for example, uncover that two reports documented on the Grain Matrix are identical with the exception that one report is defined at the Date level whereas the other is defined at the Month level. These two reports could be consolidated easily with the addition of a calendar parameter or by encouraging the business to agree on a common level across both reports, such as Date.
- **Estimates work effort.** Before starting development, you can roughly estimate the complexity of your design. The Grain Matrix, combined with the Family Tree, is a quick way to arrive at an accurate estimate for data mart development.

Exercise 11: Building The Templates

1. What role in your organization is responsible for completing the Family Tree and Grain Matrix?

At what point in the development process should these two templates be completed?

Answers

1. Someone playing the role of functional analyst should be the one responsible for completing the Family Tree and Grain Matrix. In many organizations, the role 'functional analyst' is performed by a business analyst or data modeler. These templates are often completed in parallel with the logical data modeling effort and often complete at the same time. That is, the logical data analysis phase including the logical data model, Family Tree, and Grain Matrix. Note that the Grain Matrix is only produced if we are building a reporting application.

See the Appendix for my thoughts.

Key Points

- The In-The-Know Template captures the people and documentation that can provide and validate the data requirements.
- The Subject Area List is a compilation of the key concepts the business feels are important to capture, without introducing data modeling notation.
- The Family Tree is a spreadsheet that captures the source applications and other key metadata for each subject area or data element within the scope of our application.

- The Grain Matrix is a spreadsheet that captures the levels of reporting for each subject area or measurement. It is the spreadsheet view of a dimensional model.

Chapter 12: What is the Data Model Scorecard®?

Overview

Pay now, not later
Get the data model right
Sleep better at night

A frequently overlooked aspect of data quality management is that of data model quality. We often build data models quickly, in the midst of a development project, and with the singular goal of database design. Yet the implications of those models are far-reaching and long-lasting. They affect the structure of implemented data, the ability to adapt to change, understanding of and communication about data, definition of data quality rules, and much more. In many ways, high-quality data begins with high-quality data models. Therefore, because a good data model can lead to a good application, and similarly, a bad data model can lead to a bad application, we need an objective way of measuring what is good or bad about the model. After reviewing hundreds of data models, I formalized the criteria I have been using into what I call the Data Model Scorecard®. This chapter will explain the Data Model Scorecard® and its ten categories.

Data Model Scorecard® Explained

The Scorecard is shown in table 12.1. Each of the 10 categories has a total score that relates to the value your organization places on the question. Just as in any assessment, the total must be 100.

Table 12.1: Data Model Scorecard® template

Open table as spreadsheet

#	Category	Total score	Model score	%	Comments
1	How well do the characteristics of the model support the type of model?	10			
2	How well does the model capture the requirements?	15			

Table 12.1: Data Model Scorecard® template

Open table as spreadsheet

#	Category	Total score	Model score	%	Comments
3	How complete is the model?	15			
4	How structurally sound is the model?	15			
5	How well does the model leverage generic structures?	10			
6	How well does the model follow naming standards?	5			
7	How well has the model been arranged for readability?	5			
8	How good are the definitions?	10			
9	How consistent is the model with the enterprise?	5			
10	How well does the metadata match the data?	10			
	TOTAL SCORE	100			

The model score column contains the results of how a particular model did, with a maximum score being the value that appears in the total score column. For example, if a model received 10 on "How well does the model capture the requirements?" then you would put 10 in this column. The % column presents the Model Score for the category divided by the Total Score for the category. For example, receiving 10 out of 15 would lead to 66%. In the comments column, place any pertinent information that explains the score in more detail or captures the action items on what is required to fix the model. The last row contains the overall score assigned to the model, a sum of each of the columns.

Table 12.2 includes an example of the Scorecard template filled in.

Table 12.2: Data Model Scorecard® example

Open table as spreadsheet

#	Category	Total score	Model score	%	Comments
1	How well do the characteristics of the model support the type of model?	10	10	100%	Lots of processing data elements
2	How well does the model capture the	15	14	93%	Revisit some AKs

Table 12.2: Data Model Scorecard® example

Open table as spreadsheet

#	Category	Total score	Model score	%	Comments
	requirements?				
3	How complete is the model?	15	15	100%	Legacy system mapping
4	How structurally sound is the model?	15	10	67%	Null AKs
5	How well does the model leverage generic structures?	10	10	100%	Perfect use of abstraction
6	How well does the model follow naming standards?	5	4	80%	Innovative standard for table naming
7	How well has the model been arranged for readability?	5	4	80%	Incorporate a subject area model
8	How good are the definitions?	10	9	90%	Correct and complete definitions
9	How consistent is the model with the enterprise?	5	5	100%	Strong rapport with business
10	How well does the metadata match the data?	10	10	100%	Handles changing natural account numbers
	TOTAL SCORE	100	91		

The model that was reviewed in this example received a score of 91. Category 4 is a strong candidate for improvement and categories 6 and 7 also contain areas that could be improved. It is useful to provide a document that explains the results in more detail, to accompany a completed scorecard. In the example from Table 12.2, this document was over 50 pages in length. Both strengths and areas for improvement are explained in detail through a complete set or representative set of examples. For example, Category 2 lost some points because this model had several incorrect alternate keys. In the accompanying document, those entities with suspect alternate keys are captured.

Feel free to use the Scorecard on your own projects — just please add this reference:

- *Steve Hoberman & Associates, LLC hereby grants to organizations a non-exclusive royalty-free limited use license to use the Data Model Scorecard® solely for internal data model improvement purposes. The name 'Steve Hoberman & Associates, LLC' and the website 'www.stevehoberman.com' must appear on every document referencing the Data Model Scorecard®. Organizations have no right to sublicense the Data Model Scorecard® and no right to use the Data Model Scorecard® for any purposes outside of the organization's business.*

The Scorecard has three main characteristics that make it an invaluable tool:

- **The Scorecard starts by assuming the model is perfect.** As analysts, we sometimes notice immediately what is wrong. This can lead to quickly pointing out the negatives in designs, which in turn, can make us blind to what is good in the model, causing conflict or hard feelings among project team members. The Scorecard starts off with a perfect score of 100. We then subtract points from this score for categories where we identify areas that need improvement.
- **The Scorecard is objective and externally-defined.** I have participated in model reviews where modelers take the review personally and comments take the form of "I don't like what you did here…" or "You are still not getting this structure right…" We need to step back from the 'I' and 'You', and critique the model with an external and objective perspective. Team rapport remains intact as you, the reviewer, are not criticizing their model, but rather evaluating how well the model meets pre-defined objectives, using an external scale to indicate areas for improvement.
- **The Scorecard is easy to apply and standardize.** The Scorecard was designed to enable even those new to modeling to critique their own models and the models of their colleagues. It should be incorporated into your methodology as a final checkpoint before the model is considered complete.

Let's explore each of these ten categories in more detail.

1. How Well Do the Characteristics of the Model Support the Type of Model?

This is the "type" category. This question ensures that the type of model (subject area, logical, or physical — and then either relational or dimensional) has the appropriate level of detail. In general terms, the subject area model should contain a well-defined scope, the logical data model should be application-independent and represent a business solution, and the physical data model should be tuned for performance, security, and take into consideration hardware and software. The physical data model should represent a technical solution. A dimensional model is built when there is a need to play with numbers, while a relational model is built for everything else.

This category is challenging to grade because the modeler and reviewer need to understand the definition of each model type. Also, during crunch times, one model might be created to represent both logical and physical views, for example, thus complicating the grading of this category. I call this type of model a "physio-logical" model.

Each subject area on a subject area model must be basic and critical. This means that it is core to the business being modeled, and that without this concept, the business being modeled would be very different or possibly non-existent. Customer, Product, and Employee are all examples of subject areas.

If a logical data model is relational, it should be fully normalized. If a logical model is dimensional, every hierarchy level should be shown and there should only be a single meter. It needs to be completely independent of software and hardware.

The physical data model represents context. Will the logical data model work with a given set of software and hardware? If there are opportunities to improve performance, space management, and security, for example, the model should be modified cautiously to allow applications to use it more efficiently.

For example, Figure 12.1 contains a relational logical data model that is not fully normalized — the many-to-many relationship between Employee and Department has not been resolved. Figure 12.2 contains the relational logical data model after this category catch has been fixed. A new associative entity has been added to resolve the many-to-many.

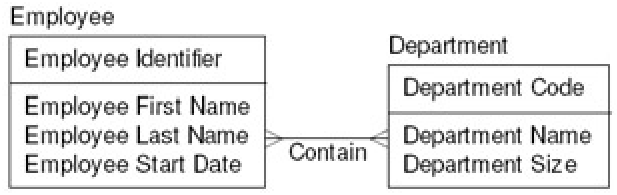


Figure 12.1: Category catch—Relational logical data model not fully normalized

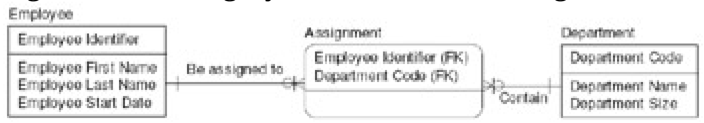


Figure 12.2: Category catch fixed—Associative entity added

2. How Well Does the Model Capture the Requirements?

This is the "correctness" category. That is, we need to understand the content of what is being modeled. This can be the most difficult of all 10 categories to grade because we really need to understand how the business works and what they want from their application(s). If we are modeling a sales data mart, for example, we need to understand both how the invoicing process works in our company, as well as what reports and queries will be needed to answer key sales questions from the business.

We need to ensure that our model represents the data requirements, as the costs can be devastating if there is even a slight difference between what was required and what was delivered. Besides not delivering what was expected, there is the potential for the IT/business relationship to suffer. The model must support business expectations.

For example, Figure 12.3 contains a dimensional physical data model for a reporting application. One of the questions defined in the requirements is "Show me Net Sales by Customer and by Quarter." The model does contain **Net Sales Amount**, but the calendar view does not contain Quarter. Figure 12.4 shows the physical data model after this category catch has been fixed by replacing Month with Quarter. Note that when confirming this requirement, we might learn that both Month and Quarter are required, instead of one or the other. We might also learn that Quarter was not added deliberately, because it could be easily derived by aggregating three Months of data.

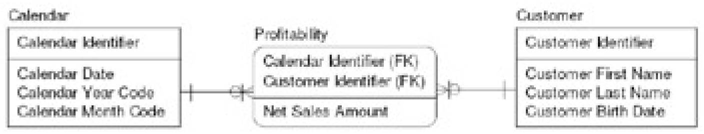


Figure 12.3: Category catch—Physical data model that does not answer the required business question

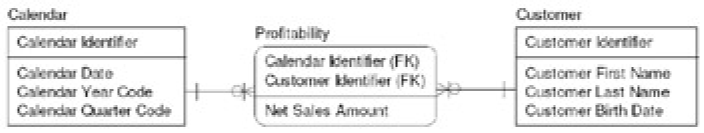


Figure 12.4: Category catch fixed—Month replaced by Quarter

3. How Complete is the Model?

This is the "completeness" category. This category checks for data model components that are not in the requirements or requirements that are not represented on the model. If the scope of the model is greater than the requirements, we have a situation known as "scope creep." There is more on the model than is specified in the requirements. Therefore, we are planning on delivering more than what was originally required. This may not necessarily be a bad thing, as long as this additional scope has been factored into the project plan. If the model scope is less than the requirements, we will be leaving information out of the resulting application, usually leading to an enhancement or "Phase II" shortly after the application is in production. For completeness, we need to make sure the scope of the project and model match.

Also for completeness, we need to ensure that all the necessary data model descriptive information (also known as metadata, which will be discussed in Chapter 16) is populated. So for this category, we need to make sure both data and metadata scope are appropriate. Regarding metadata, there are certain types that tend to be overlooked when modeling, such as definitions and alternate keys. These types of metadata, along with those that are mandatory parts of our model such as data element name and format information, need to be checked for completeness in this category.

For example, in Figure 12.5, the Promotion entity contains a surrogate key but not a corresponding alternate key. The alternate key in this case will be the natural business key. That is, what data element or data elements would make a promotion unique in the eyes of the business? Figure 12.6 contains this data model after this category catch has been fixed by confirming with the business that the natural key is both **Promotion Code** and **Promotion Start Date**.

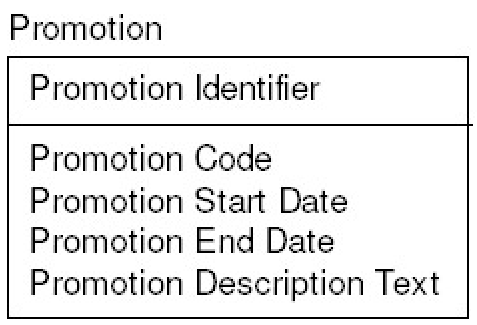


Figure 12.5: Category catch—Surrogate key missing alternate key

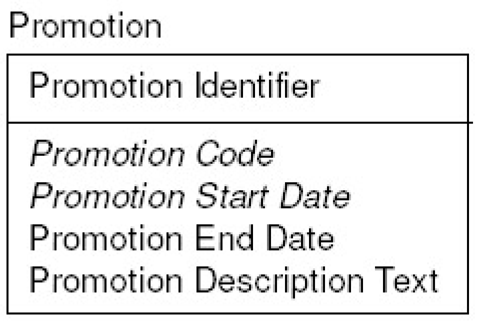


Figure 12.6: Category catch fixed—Alternate key added

4. How Structurally Sound is the Model?

This is the "structure" category. This category validates the design practices employed to build the model to ensure we can eventually build a database from our data model, avoiding design issues such as having two data elements with the same exact name in the same entity, a null data element in a primary key, and partial key relationships [1]. Traditionally, when we review a data model, the violations we catch fall into this category, because we don't need to understand the content of the model to score this category. If someone knows nothing about the industry or subject matter the model represents, this category can still be graded accurately.

Many of the potential problems from this category are quickly and automatically flagged by our modeling and database tools. Structural soundness issues that escape the human eye, or are very tedious to check for can be identified easily by many data modeling tools. For example, checking the primary keys in each entity to ensure they are mandatory and not optional.

In the logical data model in Figure 12.7, the two relationships between Customer and Account are both mandatory. The first relationship captures that each Customer may own many Accounts, and each Account must be owned by a single Customer. This means that an Account cannot be created without a Customer. The second relationship however captures that each Account may be owned by many Customers, and each Customer must own one Account. This means that Customer cannot be created without an Account. We have a circular relationship, meaning we cannot create any instances in either entity because the other entity instance must exist first. We cannot create Bob the Customer, for example, without creating his Accounts. However, we cannot open an Account for Bob without Bob already existing. So we are stuck.

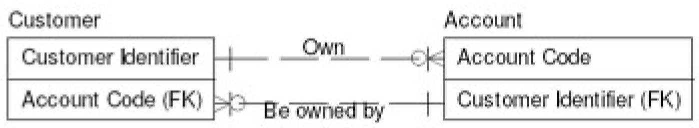


Figure 12.7: Category catch—Logical data model with circular relationship

Once we realize that this circular relationship is really a many-to-many relationship between Customer and Account, we can fix it by adding an associative entity between the two, such as what is shown in Figure 12.8.

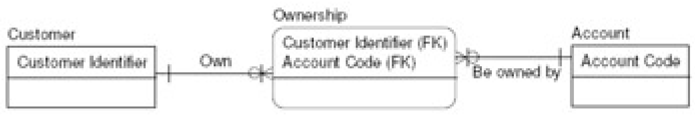


Figure 12.8: Category catch fixed—One way to resolve circular relationship

However, the circular relationship in Figure 12.7 might also be hiding richer business rules that require us to ask more business questions to determine the significance of each relationship. It is possible, for example, that the Account foreign key in the Customer entity in Figure 12.7 represents the Customer's primary account. If so, the circular relationship can be fixed, as shown in Figure 12.9.

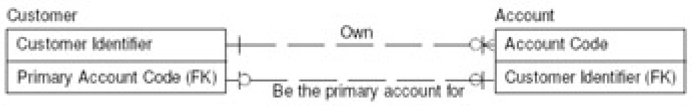


Figure 12.9: Category catch fixed—Another way to resolve the circular relationship after learning we need to capture the primary account

This model however, still raises eyebrows because the one-to-one relationship is optional on both sides, meaning that we don't have to have a Primary Account and we don't have to have a Customer.

[1]A partial key relationship is when only part of the parent entity's primary key is brought over to the child entity as a foreign key. Good design practice ensures that when a relationship is created between two entities, the entire primary key from the parent entity on the 'one side' of the relationship line is copied over to the child entity on the 'many side' of the relationship line. A partial key relationship is when a subset of those data elements in the primary key are copied instead of all of the data elements in the primary key.

5. How Well Does the Model Leverage Generic Structures?

This is the "abstraction" category. This category confirms an appropriate use of generic structures on the model. One of the most powerful tools a data modeler has at their disposal is abstraction, the ability to increase the types of information a design can accommodate using generic concepts. Recall our earlier discussions on abstraction in Chapters 3 and 9. Going from Customer Location to a more generic Location, for example, allows the design to handle other types of locations, such as warehouses and distribution centers, more easily. Abstraction can be properly applied (or abused!) at the entity, relationship, and data element levels.

This category can be challenging to grade because you need to know how to phrase "what if" scenarios and find people in the business who can answer these questions. For example, if you are considering the party role concept, and currently you only have Customer on your model, someone needs to confirm that in the not-so-distant future you will have other types of party roles, such as Employee and Supplier.

Also, you (or your management) will need to know how to weigh the potential value of abstraction with the effort required to implement an abstract structure. As a modeler, I thought abstraction was the most amazing concept. Then I volunteered to take over a development project and my first assignment was to implement an abstract structure I had designed! It was very painful and I learned that flexibility always comes with a price.

Figure 12.10 contains a subject area model whose goal is to clearly explain Customer. In its present form, it is way too abstract to explain Customer.

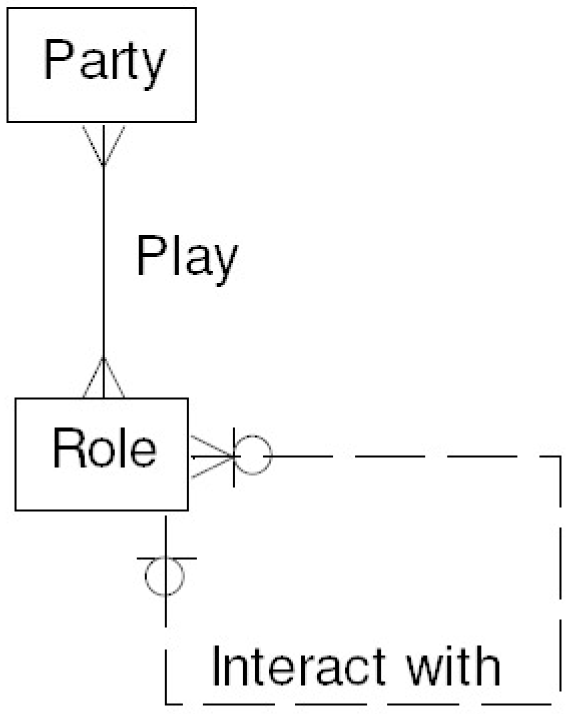


Figure 12.10: Category catch—Overly abstracted Customer subject area model

If we do need to keep Party in this model, one way to bring back Customer is through subtyping, as shown in Figure 12.11.

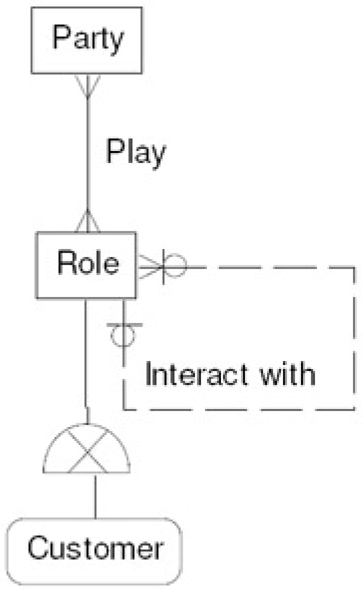


Figure 12.11: Category catch fixed—Customer is back

6. How Well Does the Model Follow Naming Standards?

This is the "standards" category. Correct and consistent naming standards are extremely helpful for knowledge transfer and integration. New team members who are familiar with similar naming conventions on other projects will not have to take time to learn a new set of naming standards. This category focuses on naming standard structure, term, and syntax.

Structure includes the components of a name. A popular standard for the data element name structure, for example, is one Subject Area, zero, one, or many Modifiers, and one Class Word. A subject area is a concept that is basic and critical to the business. A modifier qualifies this subject area and a class word is the high-level domain for a data element. Examples of class words are Name, Quantity, Amount, Code, and Date. **Customer Last Name** for example, ends in the 'Name' class word.

Term includes proper spelling and abbreviation. An abbreviations list can be used to name each logical and physical term. Organizations should have a process in place for efficiently creating new abbreviations if a term cannot be found on a list. The process should be carefully managed to prevent different abbreviations from being created for the same or similar term, and for creating the same abbreviation for completely different terms.

Syntax includes determining if the term should be plural or singular, if hyphens, spaces, or Camelback (i.e. initial upper case such as CustomerLastName) should be used, and case (i.e. upper case, initial upper case, or lower case). Note that entities and attributes are usually singular because they represent one instance in the database (similar to one row in a spreadsheet).

For example, the subject area model in Figure 12.12 contains a plural entity name. This can be easily fixed as shown in Figure 12.13.

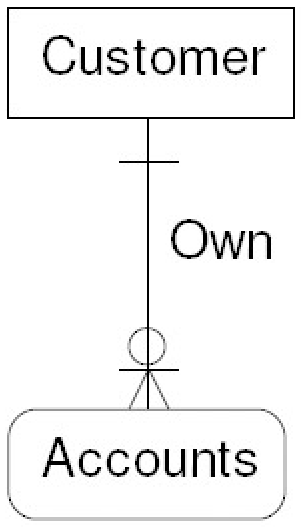


Figure 12.12: Category catch—Subject area model with plural name

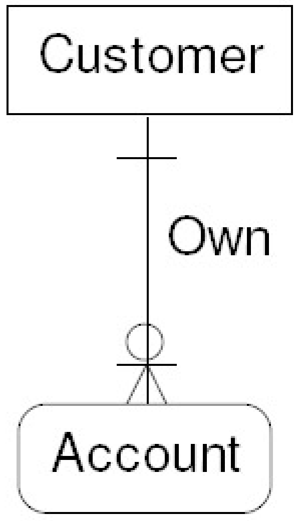


Figure 12.13: Category catch fixed—Now Account is singular

7. How Well Has the Model Been Arranged for Readability?

This is the "readability" category. This question checks to make sure the model is visually easy to follow. Readability needs to be considered at a model, entity, data element, and relationship level.

At a model level, I like to see a large model broken into smaller logical pieces. I also search for the "heart" of the model. That is, when looking at a data model, the part of the model your eyes are naturally attracted to. This tends to be an entity or entities with many relationships to other entities, similar to the hub on a bicycle tire with many spokes to the outside rim of the tire. This "heart" needs to be carefully positioned so that the reader can identify it early on and use this as a starting point to walk through the rest of the model. On a dimensional data model for example, the fact table is the model's heart and should be in the center surrounded by dimensions.

At an entity level, I like to see child entities below parent entities. Child entities are on the many side of the relationship, with parent entities on the one side of the relationship. So if an Order contains many Order Lines, Order Line should appear below Order.

At a data element level, I like to see some logic applied regarding the placement of data elements within an entity. For example, on reference entities such as Customer and Employee, it is more readable to have the data elements listed in an order that makes sense for someone starting at the beginning of the entity and working down sequentially. For example, **City Name**, then **State Name**, then **Postal Code**. For transaction entities such as Order and Claim, I have found it more readable to group the data elements into class words, such as all amount data elements grouped together.

At a relationship level, I try to minimize relationship line length, and the number of direction changes a relationship line makes. I also look for missing or incomplete relationship labels, if labels are appropriate on the model. The larger the model, the less useful it is to display labels, as the extra verbiage can make the model harder to read.

Would you believe the two dimensional data models shown in Figure 12.14 and Figure 12.15 are the same model, just arranged differently? Also, the model in Figure 12.14 contains uppercase entity names — uppercase is more difficult to read than initial upper case.

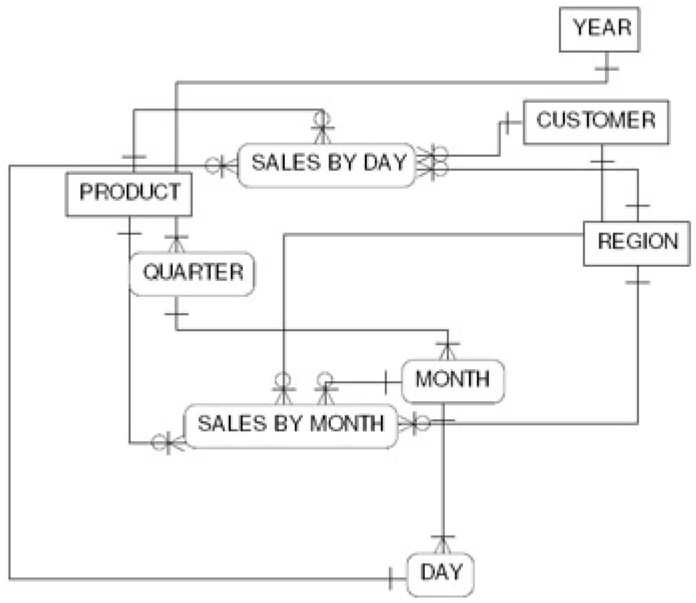


Figure 12.14: Category catch—Poor readability on subject area model

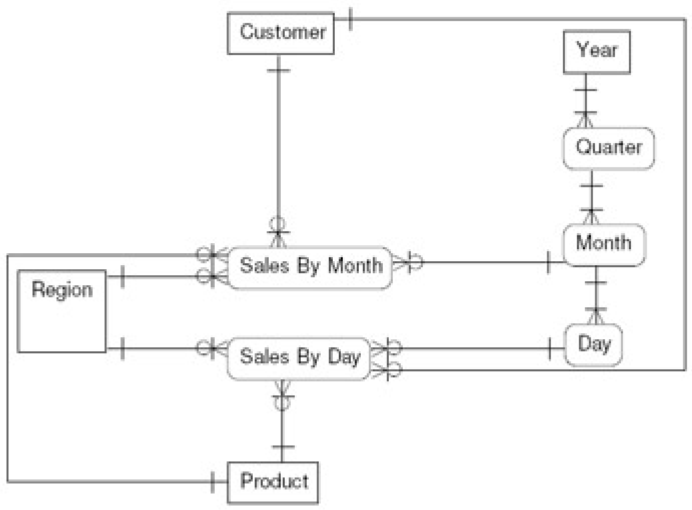


Figure 12.15: Category catch fixed—Much more readable

8. How Good are the Definitions?

This is the "definitions" category. This category includes checking all definitions to make sure they are clear, complete, correct, and consistent.

Clearness means that a reader can understand the meaning of a term by reading the definition only once. The definition is precise and unambiguous, and vague terms and cryptic abbreviations or acronyms are avoided. Also, a clear definition does not require the reader to refer to other concept definitions to fully understand it.

Completeness ensures the definition is at the appropriate level of detail, and that it includes all the necessary components, such as derivations and examples. Completeness also includes clarifying common misconceptions among similar concepts. For example, if the distinction between Consumer and Customer is not widely-known in your organization, their definitions should clarify it.

Correctness focuses on having a definition that totally matches what the term means, and is consistent with the rest of the business. Often, correctness is achieved by reusing an industry standard definition or having an acknowledged business expert "sign off" on the definition.

Consistency means the structure of the definition is the same within the organization. For example, all definitions should be present tense and captured in the singular (not plural). Definitions should start off with the same phrase, such as "A…" or "The…".

There is an international standard that provides guidance on how to formulate definitions (ISO/IEC 11179, Part 4). It is very important that definitions be captured and are clear, complete, correct, and consistent. We will explain the reasons using the example in Figure 12.16.

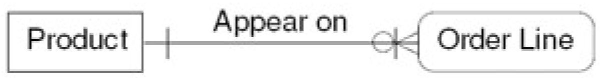


Figure 12.16: Definitions can better explain this model

Definitions are important:

- **To assist business professionals in making intelligent business decisions.** In the good old days, definitions were always behind the scenes, only to be used occasionally by technical staff when validating data elements. However, with business intelligence and data warehousing pushing data elements onto the screens and reports of business people, the definitions help confirm that the right field has been chosen. They also reduce misunderstandings. A business intelligence project can be developed flawlessly, but if a business person has a different interpretation of a data element than what was developed, it is easy for poor decisions to be made, compromising the entire application. If a business person would like to know how many products have been ordered each month, for example, imagine the poor judgments that could result if the person expected raw materials to be included with products and they were not, or if he or she assumed that raw materials were not included but they were.
- **To assist technology professionals in making intelligent development decisions.** How do we know which data element to source data from, to use in calculations, or to display to the user? We can look at sample data and the names of the data elements, but we lean most heavily on the definition. If the definition of Order Line mentions that open orders, canceled orders, and dropped orders are also within the scope of Order Line, this might influence sourcing decisions and database design decisions if space and performance become issues.
- **To support rigor on the data model.** As we can see in this example, where each Order Line must be for one and only one Product, a data model should be exact and unambiguous. An Order Line cannot exist without a Product. However, if the definition of Product is missing or vague, we would have less confidence in the entity and its relationships. Is a Product, for example, raw materials and intermediate goods, or only a finished item ready for resale? Can we have an order for a service, or must a Product be a tangible deliverable? The definition is needed to support the Product entity and its relationship to Order Line.

Figure 12.17 contains a poor definition for Customer, because it doesn't explain what the Customer is. Instead, we only learn what Customer contains. Figure 12.18 contains a clear, complete, and correct definition for Customer.

Customer contains last name, first name, and address.

Figure 12.17: Category catch—Poor definition for Customer

A Customer is a person or organization who obtains our product for resale. The Customer normally obtains the product through purchase. An example of a customer who does not obtain our product for resale is the Salvation Army, which receives the product for free as a charity organization. A person or organization must have obtained at least one product from us to be considered a Customer. That is, Prospects are not Customers. Also, once a Customer, always a Customer, so even Customers that have not purchased anything in 50 years are still considered Customers. The Customer is different from the Consumer, who purchases the product for consumption rather than resale.

Figure 12.18: Category catch fixed—Clear, complete, and correct definition of Customer

Examples:

- *Walmart*
  Bob's Grocery Store
  Military Base 1332

The reason that Figure 12.18 contains a good definition is because it is clear, complete, and correct. It is clear because even if you don't work for this company, you can clearly understand how they view a customer. It is complete because it includes enough detail so that we know, for example, how prospects and customers differ. Also, showing examples such as 'Walmart' add to the completeness of this definition. For the correctness of the definition, we will need to ensure that this definition has been signed off by somebody on the business side.

9. How Consistent is the Model with the Enterprise?

This is the "consistency" category. Does this model complement the "big picture"? This category ensures the information is represented in a broad and consistent context, so that one set of terminology and rules can be spoken in the organization. The structures that appear in a data model should be consistent in terminology and usage with structures that appear in related data models, and ideally with the enterprise data model (EDM), if one exists. This way there will be consistency across projects.

An enterprise data model (EDM) is a subject-oriented and integrated data model containing all of the data produced and consumed across an entire organization. Subject-oriented means that the concepts on a data model fit together as the CEO sees the company, as opposed to how individual functional or department heads see the company. There is one Customer entity, one Order entity, etc. Integration goes hand-in-hand with subject-orientation. Integration means that all of the data and rules in an organization are depicted once and fit together seamlessly.

Not all organizations have an enterprise data model. If no enterprise model exists, I look for widely accepted, existing models for comparison.

For example, if the model being reviewed has an entity called Credit and the closest concept to Credit on the enterprise data model is Event, how consistent are Credit and Event? Is a Credit a type of Event? We need to ensure a fit between these two concepts.

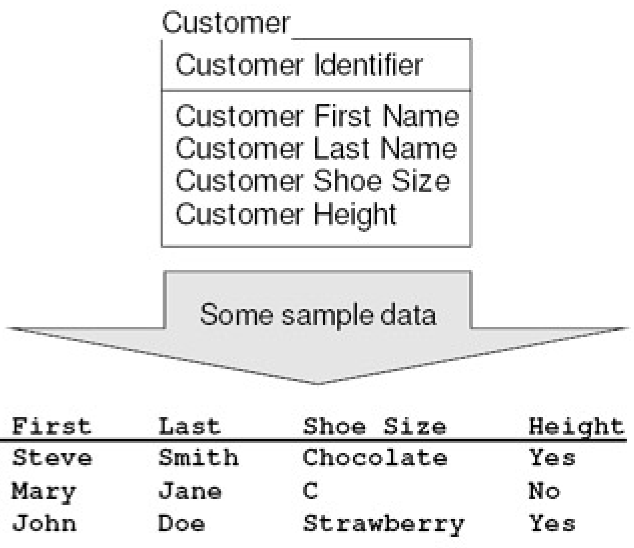
10. How Well Does the Metadata Match the Data?

This is the "profile" category. This category determines how well the data elements and their rules match reality. Does the data element **Customer Last Name** really contain the customer's last name, for example?

This category ensures the model and the actual data that will be stored within the resulting tables are consistent with each other, to reduce surprises later on in software development.

It might be very difficult to compare the data to the metadata early in a project's lifecycle because data can be difficult to access from the source system before development has begun. However, the earlier, the better to avoid future surprises, which can be much more costly. If the data model that is being reviewed is the source of the data, such as the data model for a data entry application, you will need to be creative on how to confirm the data matches the metadata. Creative techniques include examining test files, labels on screens, and studying prototypes.

For example, in Figure 12.19, look at the sample values for Shoe Size and Height. Shoe Size looks closer to an ice cream flavor than a shoe size, but what does the character 'C' mean? Height is a yes or no indicator, meaning it probably isn't really height (unless we are capturing whether something is two-dimensional or three-dimensional, in which case we probably should change the data element name).



Id	First	Last	Shoe Size	Height
123	Steve	Smith	Chocolate	Yes
432	Mary	Jane	C	No
789	John	Doe	Strawberry	Yes

Figure 12.19: Category catch—Sample data in the physical data model entity Customer

Exercise 12: Determining the Most Challenging Scorecard Category

1. Which of the Scorecard categories do you think would be the most challenging to grade, and why?

Answers

1. I believe Category 2, "How well does the model capture the requirements?" is the most difficult Scorecard category to grade. The business requirements may not be well-defined, or they differ from verbal requirements, or they keep changing usually with the scope expanding instead of contracting.

See the Appendix for my thoughts.

Key Points

- The Data Model Scorecard® is a collection of ten categories for validating the quality of a data model.
- Applying the Scorecard early in the modeling process saves rework later and increases the chances that your comments on the model will be incorporated.
- The Scorecard can be customized for a particular organization. Categories and point scores can be changed to make it work for you.

Chapter 13: How Can We Work Effectively with Others?

By Graeme Simsion

Overview

Set expectations
Stay on track, achieve closure
Strong relationships

Graeme Simsion worked in the data modeling field for twenty-five years as a practitioner, teacher and researcher. He is the author of *Data Modeling Essentials*, now in its third edition, and *Data Modeling Theory and Practice*. He now focuses on teaching consulting and facilitation skills, drawing on his experience as CEO of a successful business and IT consultancy. In this chapter, he shares his experiences of setting expectations, staying on track, and achieving closure.

Recognizing People Issues

For some years, I taught an advanced class in data modeling. In the opening session, I would ask the delegates — most of them very experienced modelers — to nominate their biggest challenge. Overwhelmingly, their responses were about working with others: persuading business and technical people of the value of data modeling; getting access to business stakeholders and communicating effectively with them; building an effective working relationship with the database administration team.

Curiously, they were then not always enthusiastic about devoting much of the class to dealing with these problems. There was a sense that they were inevitable, and that data modelers could not do much to prevent or resolve them. It's easier to stick to the technical issues.

This chapter takes the position that:

- Many of the most serious challenges you face in data modeling will be "people issues" or, if you prefer, "political issues"
- These challenges can be addressed in a disciplined way, largely by process rather than through ill-defined "people skills"
- Most modelers can substantially improve their performance in this area by adopting just a few basic principles and practices

The word *basic* in the third point deserves a comment. Much of what I cover in this chapter is "common sense", detailed in innumerable books on project management, psychology, sales, and personal development. The only originality I claim is in putting it in the context of data modeling. But as you read, ask yourself not only whether you agree with the recommended approaches, but whether you consistently apply them in your day-to-day work. The step many modelers need to take is to commit to working in a way that they know intellectually is right, but which gets lost in the pressures and emotions of the immediate situation.

This chapter falls logically into three sections: broadly, *setting expectations, staying on track*, and *achieving closure*. The first is the most important: in my experience failure to clearly establish mutual expectations is the most common reason for the failure of assignments both within and outside the data modeling sphere.

In looking at these "soft" challenges, I recognize that data modelers do more than create data models. They may be asked to review existing models, advise on modeling approaches, evaluate tools … In much of this work, they will find themselves effectively (or actually) working as consultants, providing services to a client rather than simply working to direction. Specialist data modelers may be part of a common team (an internal or external consultancy), with its own objectives above and beyond those of the project team. Consulting brings its own relationship management challenges, and we will look at these along the way.

Setting Expectations

A successful data modeling engagement needs to be based on a set of expectations that are understood and agreed upon by all stakeholders. Easy to say, but in the haste to get moving, we can forget the lessons from our programmer brethren (and indeed the advice we would give on the database side) and fail to put a proper specification of work in place. We will look at some key questions to ask — and who to ask — in establishing such a specification, but before we bury ourselves in the detail, we need to establish the big picture.

Understanding Context

It's sometimes called "asking the next-higher question." "Why are we doing this? What does the client / project manager / sponsor want it for? And, consequently, exactly what sort of deliverable is needed to serve that purpose?" I would suggest that failure to understand and address this *context* in which work is being performed is the single greatest cause of problems in data modeling assignments.

A simple example: a team was assembled to undertake a classic data mining project, looking at an operational database for "interesting correlations". The highly-qualified team included a data modeler to reverse engineer the database, as well as a statistician to interpret results. After some preliminary work, they produced a steady stream of "interesting correlations", but the client remained consistently unimpressed. At this point, I became involved. My first question to the "difficult" client was, "why are you doing this?" The response: "Because our organization has been criticized in a public report, and we wanted to know whether any of the claims in the report could be refuted by our operational data." It took no more than an hour with a copy of the report and a highlighting pen to identify the statements that might be corroborated or refuted by the data, and only a day for the data mining team to produce answers — to the satisfaction of the client.

In the review that followed, the mining team was quick to point out that the client had not explained the reason for the work, that they'd never heard of the report, and that the work actually required wasn't data mining anyway! I also suspect that having been given a brief to do exactly the sort of work they found professionally satisfying, they were not inclined to ask too many questions.

In data modeling at the project level, an understanding of context is essential in establishing quality, cost and timing requirements. The two most common complaints about data modelers are that they are impractical and that they take too long. I'm well aware of the defenses to these accusations, having used them many times myself, but we have to acknowledge the perceptions. And they are *context* issues. Behind the complaints is the suspicion that some data modelers are more concerned with the intrinsic quality of the model than with its contribution to the success of the project: "the operation was a success but the patient died".

Like other professionals, data modelers want to produce high-quality results — but need to remember that quality is *fitness for purpose* rather than an absolute standard. We are well aware of the consequences of substandard models, but are often less aware of how to compromise (and indeed are sometimes not willing to compromise) in order to meet overall project goals. If the project is time-critical, then timely delivery may be more critical than intrinsic quality factors. We should also remember that data modeling has no natural end. As with any design product, a data model can always be improved — indefinitely. It's easy to be caught in the perfection trap: pursuing the perfect "correct" model when all we are doing is refining a design beyond the point of worthwhile returns.

At least in applications development, we have a broad idea of the purpose of the data model. In strategic work, such as development of an enterprise data model, we cannot take even this for granted. Time and again I have seen enterprise models developed without a precise understanding of how they will be used. The result is frequently a model with unnecessary detail, or perhaps unfamiliar structures that do not sit easily with an organization's strategy of buying off-the-shelf software. Some advocates of enterprise models will argue that they need to be developed in "excruciating detail" - fine, as long as we have established exactly how and by whom that expensive detail is going to be used.

Remember — there is no such thing as a good data model independent of its purpose. Data modeling takes place in the context of a larger project, and will ultimately be judged by how well it contributes to that project's success.

Identifying the Stakeholders

In the next subsection, we will look at some techniques for establishing expectations in more detail. But whose expectations? A common mistake in consulting is to identify a person or team as "the client" only to find at the end of the assignment that other stakeholders' needs have not been met. The following list of potential stakeholders and some of their likely expectations is a useful starting point.

- The *database designer* is a key "hands on" user of the final model. Many times I have seen data models developed without any serious preliminary discussion with the DBA. The model is handed over on the due date — and then the arguments start. The model is too generic, too hard to understand, won't perform. Setting — and arguing — the evaluation criteria at the outset, before substantial work has been done, can substantially reduce this risk.
- The *process developers*, from business analysts and process designers to programmers and testers, will need to be able to understand and work with the model. As in the case of the DBA, we need to understand what they expect of the model.
- More broadly, the *users* of whatever we are producing — be it an enterprise model, a review, a reverse-engineered database. We need to identify, at least at a group level, those who will be affected by any change that will result from our work.
- The *project manager* and *sponsors*, who have authorized the funding of the total package of work or the data modeling component, will have their own expectations. Our basic questions to them are: Why have you asked for this? How do you plan to use it? Budgets and timelines may have some flexibility — or not. And the data modeling component may be seen as worthy of additional effort to get the best result — or it may not.

- *Subject matter experts* who will be concerned about the time required of them and their staff.
- *Technical stakeholders* are people outside the project who require adherence to standards — perhaps use of particular tools or platforms. External consultants are likely to have to meet the expectations of a purchasing department, as embodied in a legal contract.
- The *consultancy manager* is typically your line manager — perhaps the head of a central data modeling or enterprise architecture group. They may have their own agenda and expectations — typically, that work conforms to enterprise standards and that documentation is added to a central repository. If you're working for an external consultancy, there is likely to be an agenda of building a reputation and relationships that will lead to further work.

There is nothing novel in the above analysis of stakeholders and needs; the challenge is to act on it and actually establish these and any other stakeholders' needs at the outset of the exercise. This is also a good time to think about the level of involvement that these players should have in the work. One of the great lessons of change management is that people will work very hard to implement something that they have helped to design — and conversely will be resistant to ideas and designs imposed upon them. Perhaps it's worth involving that "difficult" DBA in the modeling effort — right from the start.

Asking Key Questions

At the outset of a data modeling assignment, we need to establish the expectations of all of the stakeholders (clients). We recognize that these may change as work proceeds, and will commit to monitoring and documenting such changes. Our motto should be "no surprises".

Here are a few questions that can help clarify expectations, both in terms of deliverables and method of working.

- **What does the final product look like?** The client should understand *in concrete terms* what they will be getting. Words like *architecture, model, strategy,* and even *report* are ambiguous, and we can have no confidence that a client who signs up for one of these has the same thing in mind as us. If we can't show them the deliverable from a previous exercise as a sample, we will need to generate a sample in an early project phase. Until the client has seen and agreed to a concrete example, we cannot claim to understand their expectations. In the case of a report, it makes sense to outline section headings and approximate number of pages as indications of depth and effort.
- **What does "success" look like to the stakeholder?** Sometimes the answer, to our surprise, is very different from our understanding, although the stakeholder is unlikely to tell us directly. You might be brought in to "improve" or "coach" a team, or review their performance, sponsored by a more senior manager. The team or individuals may take a defensive position, and aim to demonstrate that their current practices could not be improved upon - "success" in their terms.
- **Who will direct the work day-to-day?** Is the client interested only in the deliverable, or do they want to direct the process? This is likely to affect financial arrangements — fixed price or "time and materials".

- **Who is formally authorized to accept the deliverable?** Most of the time, one or more of your stakeholders will formally accept the deliverable. Make sure you know who this is and have them involved in the process as much as possible to ensure this signoff is a formality without any surprises.
- **How will disputes be resolved?** If we can envisage (say) the DBA refusing to accept our model (hopefully something we establish before the deadline), we should discuss the possibility in advance and agree on the approach. We can then work with that in mind, and if a dispute does arise, the client will at least be reassured that we anticipated it.
- **Whose name will go on the deliverable?** As a consultant, I encourage the client to own the deliverable with my role being to help them develop it. This makes handover much easier — and makes rejection a great deal less likely!
- **What provision will be made for follow-up?** Particularly if the client will be charged (internally or externally) for further work, it is much easier to sell this at the outset, when it is likely to be perceived as responsible planning, than at the end, when it may be seen as an attempt to secure further work — possibly requiring new expenditure approvals.

Packaging it Up

The end-result of this initial establishment of expectations — typically taking a week or two — should be a detailed plan for your component of the work, integrated, of course, with the project plan. It is critical that this becomes the *sole* documentation of expectations — overriding or specifically incorporating any previous promises. Many assignments begin with a conversation and an informal agreement — and clients have been known to bring such conversations up at the end of the assignment.

The plan should include a schedule of interviews. Many an assignment has been held up by key people being unavailable. If this happens, we want to have the leverage (or at least excuse) of a written commitment.

Staying on Track

Having established expectations, our prime task is to devote our efforts to meeting them. Note that I say *meeting* rather than repeating the conventional wisdom that we should aim to exceed expectations. Generally, exceeding expectations means doing more than what we agreed to do — and someone has to pay for that. If you think that more should be done to create an acceptable product (and this is not an uncommon situation) then raise the extra work with the client and seek to have it included in the agreed plan. If the client isn't prepared to sign up for it, advise them clearly of the consequences — then get on with delivering what the client wants. As a participant in my class observed "most of the time, just meeting expectations exceeds expectations".

As work proceeds, we also need to keep our eyes open for any changes to expectations, and deal with problems as they arise. In this section, I suggest a few practices that can help keep you on track, and some approaches to dealing with problems — in particular "people problems".

Following Good Practices

In the spirit of Stephen Covey's *Seven Habits of Highly Effective People*, here are seven "habits" or practices that can help keep an assignment on track.

- **Habit 1: Work close to the client.** We often have a choice as to where to locate ourselves: at the client's place of work, back at our "home" desk, or at our real homes. Whenever possible, I choose the client's place of work, even though it may be less convenient. Clients like to see us working — some will assume that if we're not on site, we'll be distracted by other tasks. I can keep my ear to the ground, test ideas informally, and build some relationships with the client team. If I want to build joint ownership of the project and the deliverables (and I generally do!), then being physically close makes it much easier.
- **Habit 2: Keep in touch with all of the stakeholders.** We need to stay in touch with all stakeholders, in particular the sponsor and technical buyers, to ensure their expectations are being met and have not changed. Unless we explicitly schedule meetings, chances are we will ignore the people who are not providing direct input to our task — until the end of the assignment, when they appear from the shadows. "Sure", we say to the sponsor, that's what we originally agreed, but we've persuaded the design team to move a long way from that…" Unfortunately, the sponsor has not made the journey with us.
- **Habit 3: Support the high-level relationship.** A key factor in the perceived success of a consulting assignment is the quality of the highest level relationship between the consulting organization and the client organization (which may of course be divisions of the same business). In other words, your boss should be in touch with the client's boss. It's your job to facilitate this; tell your boss when the time is right for a lunch or a coffee and keep him or her up to date so that a phone call from the client's boss doesn't come as a rude surprise.
- **Habit 4: Organize real progress meetings.** A key goal of regular progress meetings should be to "sign off" work performed so far, so that we eliminate the risk of being off track for a significant time. When the client explodes "this isn't what we asked for!" I want to be able to say "At last week's progress meeting we agreed we were on track, so at worst we've wasted a week…" Achieving this isn't simple — but it's possible.
- **Habit 5: Listen.** When I ask other consultants to share their most important lessons, much of their advice boils down to "listen". As experts and / or consultants, we are tempted to keep justifying our presence by offering advice, "adding value", and generally pontificating, when what the client wants is help taking their own plans and ideas forward. Salespeople will tell you that they have to let the potential client speak, because not speaking means not buying. The same applies to selling ideas.
- **Habit 6: Take time out.** Stephen Covey would say "sharpen the saw", based on the old story about the lumberjack who was too busy cutting down the tree to do that. But there's more to it than just improving your skills. If you are working eighty hours a week, I guarantee you will have lost perspective. You can do without that for a little while (but make sure you get help with any decisions), but as a specialist, it is critical that you can see your work in context and offer objective advice. Consultants should not spend all their time with one client: there are past clients to follow up, prospective clients to woo, a "home" team to stay in touch with. Most importantly, you need time out to reflect on your

experiences, on what has worked and what hasn't, and how you can incorporate the learning into your future consulting behaviour.
- **Habit 7: Keep a diary.** At the end of every consulting day, I write notes for myself, including working hours, key events, and reflections. It forces me to reflect on my work, supporting Habit 6, when the pressures of the consulting day are over. Occasionally a client will ask "what have you been doing for the last month?" and occasionally I'll ask myself the same question! The diary usually provides a reassuring answer.

Dealing with Problems — and Problem People

Inevitably, problems will arise — technical problems and people problems. Regardless of the nature of the problem, there are some general principles that can be applied:

- **Establish whose problem it is — or at least who is accountable for solving it.** If it's someone else's problem that affects your work, focus on finding ways of working around the problem rather than taking it on yourself — and getting in the way of the person responsible.
- **Take time out.** Having gained an understanding of the problem, and shown your willingness to help, find some space to reflect on it, away from the pressure and emotion of the situation — especially if you're responsible for solving it.
- **Keep it in perspective — and help everyone else to do the same.** As an expert, you are expected to be professional. Keep emotions out of it, terminating angry exchanges if necessary or just staying cool. Think about the impact at the next higher level — and discuss it with someone operating at that level. A year or two down the road, you'll wonder why you bothered losing sleep.
- **Get help.** Just explaining the problem to a peer can help put it in perspective — and of course may put you on the way to a solution. If you are part of a central team or consultancy, discuss it with your manager. As a consulting manager, I used to tell consultants that the greatest consulting sin was failure to put your hand up and ask for help.
- **Assume rationality until proven otherwise.** When people are behaving in a way we don't like, a natural inclination is to put it down to personality defects. Unfortunately, we are most unlikely to be able to change personalities. In my experience, however, most behavior in the workplace has a rational basis in furthering the person's goals. If you understand these goals, you can often work out what is going on. Or, working in the other direction, you can ask "under what circumstances might a rational person do this?"
- **Know your own sensibilities and vulnerabilities.** Most of us have "buttons" that are pushed by certain people or circumstances. "Know thyself" is the watchword here, so we can recognize when the problem is (at least partly) ourselves. If you know your (Jungian-based) Myers-Briggs type [1] and its implications, you may find yourself smiling when you recognize a clash that can be explained by that model. For example, Types N and S clash over levels of detail; Types J and P over the need for closure.
- **Keep the focus on the path forward.** When things go wrong, it's natural to look for someone to blame — especially if you're confident it's someone else's fault. I strongly suggest that if you must do this, you leave it until the problem is solved — which is

usually the greater priority. And once it is solved, the actual impact is known, replacing fear of unbounded consequences.
- **Keep it in the kitchen.** Keep any disputes that arise within your "home" team within that team, just as, if you were hosting a dinner party, you would not expose your guests to arguments about the cooking.
- **Learn from the experience.** When the problem has gone — solved, bypassed, perhaps ignored — convene a meeting with at least one other person, ideally from your "home" team, and reflect on the experience. Lessons learned from mistakes are lessons remembered, so it's worth making the most of them.

[1]Myers, Isabel Briggs with Peter B. Myers (1980, 1995). *Gifts Differing: Understanding Personality Type*. Mountain View, CA: Davies-Black Publishing. ISBN 0-89106-074-X.

Achieving Closure

Most of us have had the experience of working hard to get a model or report completed on time, and feeling more than a little anxious about how it will be received. Its presentation becomes a big event: will the work be accepted as it stands, will the client want changes, or will they take a deep breath and begin, uncomfortably, "you've obviously put a lot of work into this, but…"?

In a well-managed assignment, this situation should not arise. The final deliverable should be just one more step in a journey that all stakeholders have taken or at least followed. If there are problems, they should be with the last increment of work.

We can do a few things to facilitate a tidy closure. Not surprisingly, most of them need to be done throughout the assignment rather than at the end.

If we can define the end of the assignment as a hand-over rather than a sign-off, the question to the client changes from "do we need to do more to make it right?" to "are you able to take it from here?". Practically and psychologically, it will be easier to get a "yes" to the second option.

If our working dynamic is that we are helping the client develop something, rather than doing it for them, acceptance will accrue on a day-by-day basis rather than as a handover or sign-off.

Rather than a single sign-off, we should schedule staged reviews — and make it clear that if stakeholders change their minds, they will incur cost and time penalties. Expectations, as documented, should be met progressively rather than with a "big bang" deliverable.

If we are going to introduce change or (especially in the case of data modeling) unfamiliar concepts, we need to take other stakeholders along with our thinking as it progresses, rather than presenting the accumulated result, and requiring them to accept it in one leap.

Writing Reports

The tangible outcome of many assignments is a report. And the most common problem with reports is a focus on impressing rather than informing. Blame it on school! There, if we wanted good marks, our essays and assignments needed to demonstrate effort, originality, literary merit, and of course knowledge. In a business or technical report, these attributes become flaws. In particular, including material that the reader could be expected to know already, or which is not directly relevant to the assignment is likely to be seen as evidence of unnecessary effort and expense.

Re-check the expectations and write plainly and simply to address them. It can be helpful to start by writing a one page summary at the outset, as an aid to thinking about structure, then re-drafting it at the end. In any event, all reports should include such a summary, and, as with the rest of the report, stakeholders should have an opportunity to review and contribute as it develops.

Following Up

On completion of a piece of work, it's easy to lose touch with the ongoing project. The model is complete and in the hands of the DBA; new challenges beckon; perhaps the project is one you'd rather not remember. Despite these temptations to do otherwise, you should make a practice of staying in touch with past clients. At the very least it shows that you had (and have) a commitment to the project rather than just the model — remember the importance of context!

There are other reasons for staying in touch in a formal, scheduled way, as well as for occasional informal catch-ups.

Follow-up reviews provide an opportunity to correct errors and misunderstandings. Perhaps the data model needs modification (as distinct from "fudging"!) to cope with an overlooked requirement; or the definition of an entity is being misinterpreted.

Ideally, a post-engagement review should be undertaken by your line manager with the key client, to provide feedback on your performance and input to your overall performance review.

By monitoring the way our work is used, we can assess its value in the context of the project or organization, and draw lessons for future work. Strategic work is seldom expected to provide immediate payoffs, but we have a professional duty to assure ourselves that it delivers the intended longer-term benefits. If it does, we have evidence that supports using the approach again. If not, we are bound to reconsider what we are recommending.

And remember, if things go wrong, blame often falls on those who have departed. By keeping in touch, you may avoid being included in that group.

Continuous Improvement

Finally, a word on continuing to improve your ability to work effectively with others as a specialist and / or consultant. I suggest you develop the habit of observing your own interactions with experts and service providers from other fields — regular providers such as your physician, accountant, financial advisor, and travel agent, as well as one-off encounters. Look for what works and what doesn't: effective means of delivering advice, communicating, resolving problems, and less-effective approaches. Then, reflect on how you can adapt them to your own work.

Exercise 13: Keeping a Diary

Keep a diary for one month. Follow Graeme's lead, and at the end of every consulting day, write notes for yourself, including working hours, key events, and reflections. Include notes on any interactions with service providers where you were the client. Is there any behaviour that you can adapt — or avoid? Has this diary proven useful? Would you consider writing in your diary on an ongoing basis?

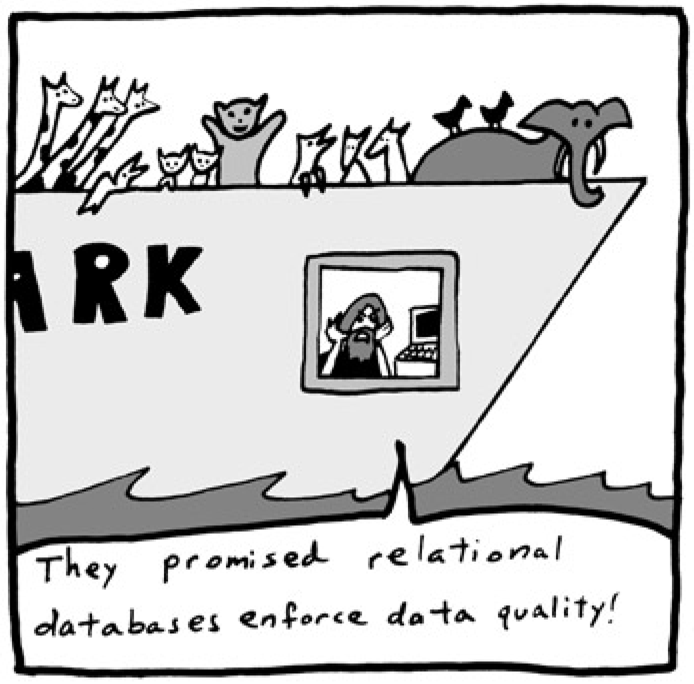
Key Points

- The biggest challenge for most data modelers is working with others: persuading business and technical people of the value of data modeling; getting access to business stakeholders and communicating effectively with them; building an effective working relationship with the database administration team.
- To improve your working relationships with others, *set expectations*, *stay on track*, and *achieve closure*.
- A successful data modeling engagement needs to be based on a set of expectations, understood and agreed by all stakeholders.
- Like other professionals, data modelers want to produce high-quality results — but need to remember that quality is *fitness for purpose* rather than an absolute standard.
- Asking questions such as "What does the final product look like?" can help clarify expectations.

Section V: Essential Topics Beyond Data Modeling

Chapter List

Part Overview



Section V introduces essential topics beyond data modeling. In Chapter 14, Bill Inmon explains unstructured data. Handling unstructured data is fast becoming part of our requirements, so we need to understand it better, as well as taxonomies and ontologies. In Chapter 15, Michael Blaha provides an overview of the Unified Modeling Language (UML). We conclude the book with Chapter 16, where I will address the top five most frequently asked questions in my classes.

Chapter 14: What is Unstructured Data?

By Bill Inmon

Overview

Text, music, pictures
Turning my world upside down
Taxonomy, help!

Bill Inmon has written 50 books, translated into 9 languages. He is considered to be the father of data warehousing. In this chapter, Bill explains unstructured data, along with taxonomies and ontologies.

Unstructured Data Explained

For years, the information technology community has focused on structured data. Structured data is data that is repetitive. There are many forms of structured data —

- bank activities
- insurance premium payments
- airline reservations
- order processing
- manufacturing job completion, and so forth.

In structured processing, the same activity is measured over and over. The only thing that changes from one activity to the next is the particulars of the activity. I go to my bank and cash a check. You go to your bank and cash a check. The only difference between my activity and yours is the date of the transaction, the amount of the transaction and the account that the transaction affected.

With structured data, the same type of activity repeats itself almost endlessly.

Indeed, even standard database management systems are optimized for storing structured data. One database record captures the information about one event and the next database record captures the particulars of the next event. Entire methodologies and disciplines have grown up around the processing of structured data.

But there is another important kind of data that is not structured. There is textual or unstructured data. It is estimated that there is approximately 4 to 5 times as much unstructured data as there is structured data in the average corporation.

Textual or unstructured data fits no pattern and is not repetitive. As a simple example of unstructured data, consider email. When a person writes an email, there is no one editing it. The email can be short or long. The email can be in English, Spanish, or Swahili. The email may contain foul language. The text in the email may be in complete sentences or not. The word "thanks" may be abbreviated "thx". An individual who writes an email is free to write whatever he or she wants. There just aren't any rules for emails. Or reports. Or a thousand other forms of textual data.

Interestingly, some of the most important information in the corporation is in the form of text. There are literally thousands of forms of textual data that contain vital information for the corporation. And yet textual data is nowhere to be found in the corporate decision making process. Figure 14.1 shows that there are two very different kinds of data in the corporation — structured data and unstructured data.

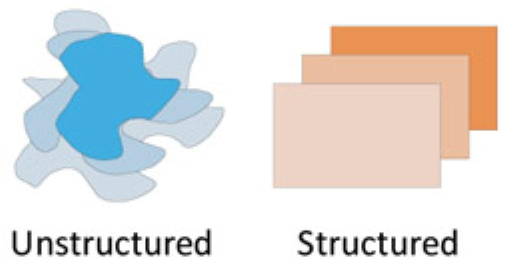


Figure 14.1: Two kinds of data in the corporation

The really good news is that there is now technology that can be used to read, integrate, and incorporate textual data into a standard relational database. In doing so, unstructured data can be incorporated into the corporate decision making process. The journey to incorporating unstructured data into corporate decisions begins with the abstraction of text.

Data Modeling and Abstraction

The notion that systematization of data begins with abstraction comes from data modeling. For a long time now, system developers have used data modeling to get a grip on structured data. It is the data model and the abstraction behind the data model that allows large types of data to be compared, to be grouped, to be treated in a common fashion. Figure 14.2 shows that the data model becomes the link between structured data and the "real world".

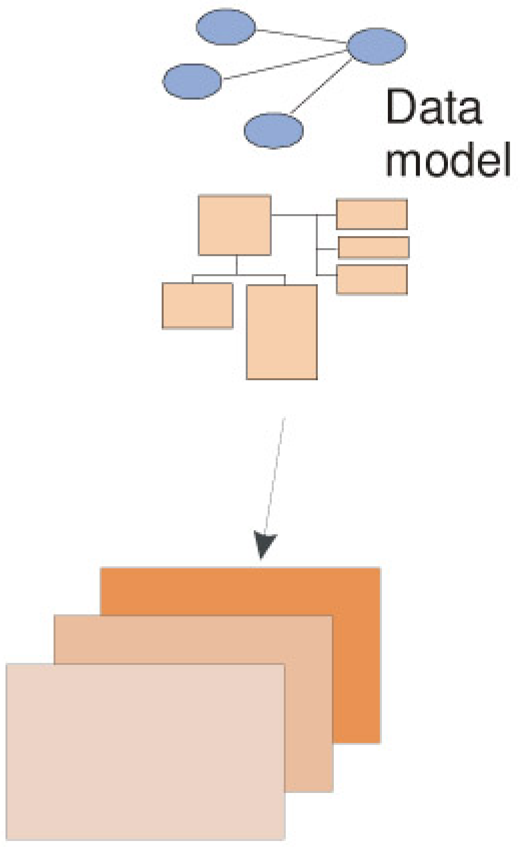


Figure 14.2: The data model is the link

The real world is usually the environment in which the system that is being modeled must operate. Typically, the real world consists of entities such as customers, products, payments, orders, transactions, shipments, and so forth. The real world is used as a basis for determining whether the data model that is based on an abstraction of the real world is "right" or "wrong". The more accurately the data model reflects the real world, the better.

The data model is useful for many purposes. It is useful for communications. It is useful for coordinating the development efforts of different groups of people over a lengthy period of time. It is useful for determining the overlap of information of different aspects of the real world. In short, the data model has many beneficial uses throughout the life of the systems that are being developed.

Modern system developers have learned that it is not advisable to build large and complex systems without a data model.

Immutable Unstructured Data

There is one underlying assumption that data modelers make about data models and the systems that are built beneath them. That assumption is that if there is a need to change the data model, the data model can be changed. Subsequently, the system that is to be built based on the data model can likewise be changed. In other words, if the government changes its mind about the length of the postal code, then the size and the format of the postal code can be increased in the data model and in the systems that are patterned after the data model. Most data modelers have taken this aspect of data modeling for granted. They simply assume that that is the way that data models and systems relate to each other. Figure 14.3 shows this aspect of the data model and the underlying systems.

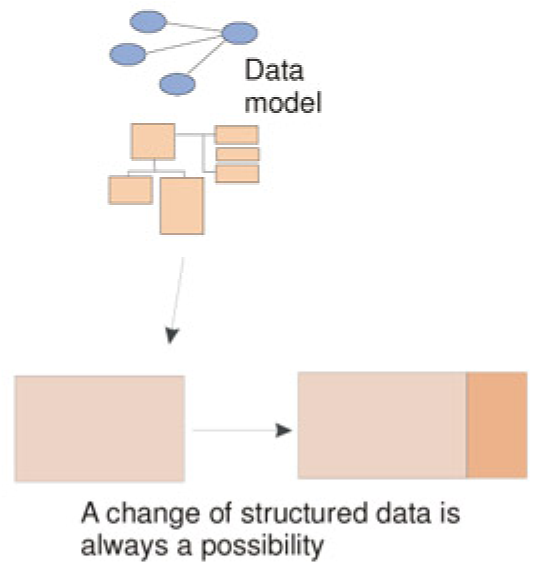


Figure 14.3: Business changes lead to data model changes

When it comes to unstructured data, such an assumption is not realistic. Unlike data found in a structured system that is mutable, most text is not mutable. In other words, text cannot be changed once written. In most cases, this is just common sense. But in some cases, this aspect of the inability to change text fringes on illegality. For example, on a loan application, a bank is charged, by law, to capture whatever the applicant has written, even if what is written is known to be incorrect. As a simple example, a person supplies their birth date as occurring 5,000 years ago. The truth is that no one lives to be 5,000 years old (Methuselah notwithstanding). But the bank is obligated to capture what was written, even when what was written is known to be incorrect.

Once it has been written, most text cannot be changed. This feature of text is diametrically different from standard structured data found in a standard system.

Taxonomies Explained

The mechanism used to abstract information in text that is similar to a data model is a taxonomy. In its simplest form, a taxonomy is merely a list of related words. There are many, many forms of taxonomies in the real world. Some simple examples of taxonomies might be —

- Cars
  - Porsche
  - Ford

    - Volkswagen
    - Honda
    - Kia, and so forth
- Or States
    - Texas
    - New Mexico
    - Arizona
    - Utah
    - Colorado, and so forth
- Or Games
    - Football
    - Hop scotch
    - Tag
    - Basketball
    - Hockey, and so forth

In its simplest form, a taxonomy is merely a categorization of some words. Each word in the category has the same relationship to the category type as every other word. See Figure 14.4.

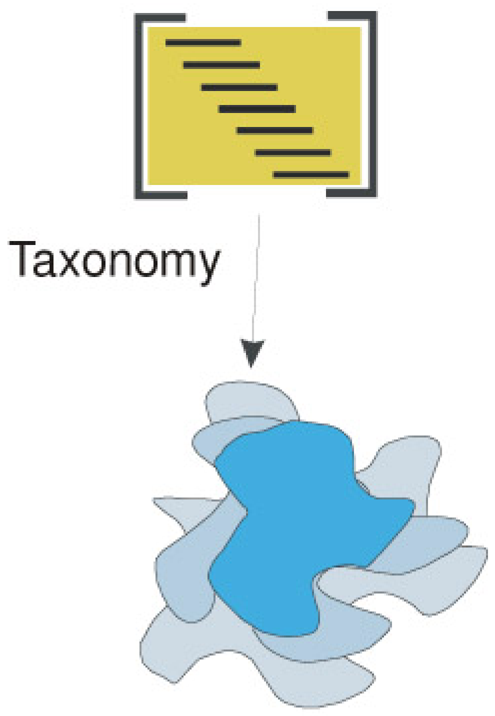


Figure 14.4: A taxonomy can be used to classify the text found in unstructured data

As an example of how a taxonomy might be used, consider some simple raw text. The raw text looks like this:

- "The young man loved to drive his Porsche. Whenever he passed a Ford, he felt a rush of adrenalin. But when he passed a Volkswagen, he felt sorry for the owner and considered it to be no special feat. Then one day he saw the new sporting model made by Honda. His insides churned."

When the reader addresses the text, the reader immediately understands the meaning of Porsche, Ford, Volkswagen, and Honda. But the computer attaches no special meaning to these words. However, when using a taxonomy against the raw text, the different occurrences of cars can be recognized. For example, by using the car taxonomy, the raw text can be transformed into:

- "The young man loved to drive his Porsche/CAR. Whenever he passed a Ford/CAR, he felt a rush of adrenalin. But when he passed a Volkswagen/CAR, he felt sorry for the owner and considered it to be no special feat. Then one day he saw the new sporting model made by Honda/CAR. His insides churned."

Processing Raw Text

By using a taxonomy, the raw text can be transformed into text that has values and categories recognizable to the computer. Using a taxonomy, the computer can now recognize what is and what is not a car. Now a query looking for all cars can be performed, and the search will find Fords, Porsches, Hondas, and so forth.

The value of the taxonomy applied to raw text is inestimable. One value is the ability to mitigate the effects of terminology. When any amount of text is gathered, it is almost inevitable that the text contains differences in terminology. For example, in medicine there are at least 20 ways to say "broken bone". Using a taxonomy, the different ways in which to say broken bone are synthesized into a common vocabulary. Once the taxonomy has been applied to raw text, the effects of terminology are mitigated.

And there are many more advantages of applying a taxonomy to raw text.

In many ways, a taxonomy is to text what a data model is to the data found in a structured system. Figure 14.5 shows this analogy.

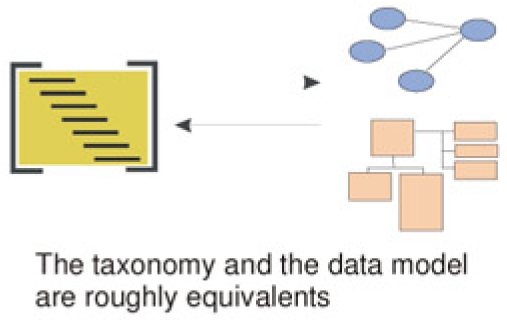


Figure 14.5: Taxonomies and data models are similar

See Figure 14.6.

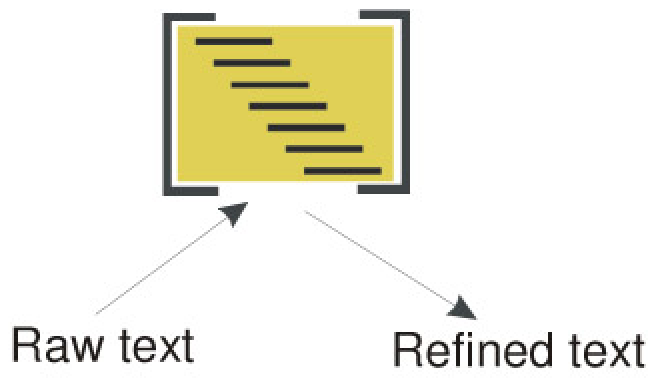


Figure 14.6: Taxonomy can produce refined text from raw text

There are many different ways to apply taxonomies to raw text. The simplest way to apply a taxonomy to raw text is to read a word of raw text and search all the words found in the taxonomy. Then, if there is a hit, apply the taxonomy classification to the raw text. Using the example that has been developed, the raw text contains the word "Porsche". The word "Porsche" is found in the taxonomy for "cars". Because a hit has occurred, the word "cars" is attached behind the word "Porsche". The result is text that looks like — "Porsche/cars". Once this attachment is made, the system can be queried looking for all references to "cars".

Note that there is a drawback to this approach. That drawback is performance. If there are n raw words to be processed and if there are m words in the taxonomy, then n x m comparisons must be made. Even at electronic speeds, this simple equation can become a stumbling block to efficient processing of raw text.

This performance consideration is especially relevant in light of the fact that there is usually more than one taxonomy to be applied to raw text. For example when processing raw text there may be taxonomies for:

- cars
- gasoline stations
- rest stops, and
- auto repair shops.

Each word in each taxonomy is used in the processing of raw text. Therefore, when one considers the simple n x m equation, one must factor in ALL the words in ALL taxonomies when calculating m.

See Figure 14.7.

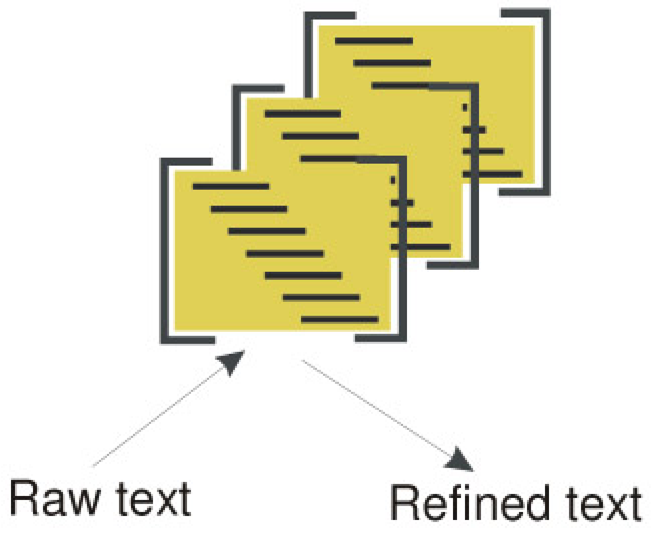


Figure 14.7: More than one taxonomy is applied to raw text in the normal case

There is another important consideration when applying a taxonomy to raw text. The taxonomy must be chosen according to its appropriateness to the raw text. To illustrate this factor consider the raw text:

- "President Ford drove a Ford"

If the raw text that was being analyzed were about American presidents, then the analysis might look like:

- "President Ford/38th president drove a Ford"

But if the raw text that was being analyzed pertained to automobiles, then the application of the taxonomy to the raw text might look like:

- "President Ford drove a Ford/car"

In light of the appropriateness of a taxonomy to a body of raw text, consider the following. Suppose the raw text being analyzed related to salt water fishing. Some of the taxonomies that apply might be:

- types of fish
- fishing methods
- oceans of the world
- deep sea oil platforms.

But it is very unlikely that the following taxonomies would be applied to the text for deep sea fishing:

- Sarbanes Oxley classifications of data

- National Football league teams
- Top 100 professional golfers
- Farming methods in the Ohio River valley.

Capturing Taxonomy Properties

Taxonomies themselves have some interesting properties. One interesting property is that of the number of words in the taxonomy. A taxonomy can contain only a few selected words or it can contain lots of words. In addition, the relationship of the taxonomy to the classification of the taxonomy may vary slightly from word to word. As an example, consider the taxonomy:

- Car
    - Porsche
    - Ford
    - Volkswagen
    - Honda
    - Humvee
    - Jeep

While it is true that each of the vehicle types listed are indeed cars (or at least vehicles), there is nevertheless a difference between cars. A Porsche is normally thought of as a sports car. A Ford is normally thought of as a family car. A Jeep is thought of as an off road vehicle.

But there are other differences. A Honda is thought of as a Japanese car. A Volkswagen is thought of as a German car. A Ford is thought of as an American car.

A taxonomy, then, is a classification of just one aspect of the objects being classified together.

Another property of a taxonomy is the fact that different levels of classification may exist within a taxonomy. For example, consider the following taxonomy:

- Car
    - Sports car
        - Porsche
        - Ferrari
    - Off Road
        - Jeep
        - Humvee
        - Mitsubishi
    - Family
        - Honda
        - Ford
        - Chrysler

It is entirely possible to have different levels of classifications or different levels of classifications within different levels of classifications. In fact, there can be all sorts of classifications within classifications. This leads naturally to the fact that it is entirely possible to have recursive relationships within a taxonomy. As an example of a recursive relationship within a taxonomy consider the following taxonomy —

- Car
    - Sports car
        - Porsche
        - Ferrari
    - Off Road
        - Jeep
        - Humvee
        - Mitsubishi
        - Porsche
    - Family
        - Honda
        - Ford
        - Chrysler

In this taxonomy, Porsche appears twice. Porsche appears as a sports car because of its world famous 911 series. But Porsche also appears in the off road classification because of its Cayenne product. See Figure 14.8.

Figure 14.8: Recursive relationships can appear within classifications themselves

Recursive relationships are interesting unto themselves. When a recursive relationship appears, interesting things can happen. Care must be taken because they require special programming attention. For one thing, programmers have to be careful because it is very easy to fall into the trap of very complex coding (that is almost impossible to debug), or loops that repeat themselves endlessly (infinite loops).

Maintaining Taxonomies Over Time

Another issue that occurs with taxonomies is that they must be maintained over time. As the world changes, taxonomies must also change. For example, in 1945, would Honda have been included in a list of cars? The answer is no. In 1990, would George W. Bush have been included in a list of presidents? The answer is no. In 1950, would Kazakhstan been included in a list of countries? The answer is no.

As the world changes, so change the taxonomies. This means that periodically, taxonomies need to be maintained. See Figure 14.9.

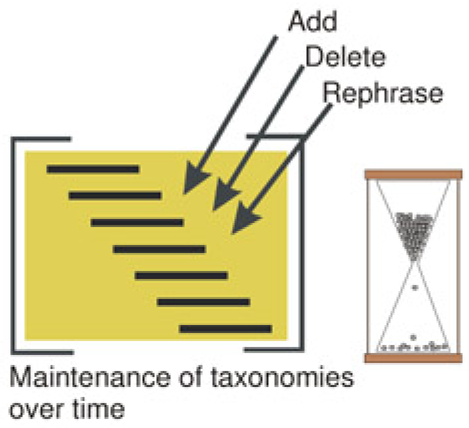


Figure 14.9: The need for periodically maintaining taxonomies

The fact that taxonomies periodically need to be maintained brings to the surface an interesting question - suppose raw text has been refined by being passed against a taxonomy. Then one day, the taxonomy is updated. Does this mean that all text that has been refined must now be reprocessed against the updated taxonomy? Or is it sufficient to merely process all future raw text against the updated taxonomy?

The answer is that on a case by case basis, raw text can be refined or not. There is no generic right or wrong here. Stated differently, some raw text will, indeed, need to be reprocessed when the taxonomy in maintained, while other raw text will not need to be reprocessed when the taxonomy is updated.

It is easy to see how simple taxonomies can be built and can be used. What may not be obvious is that taxonomies may consist of not only words, but phrases as well. For example, there may be a taxonomy of phrases that looks like Famous Hollywood lines:

go ahead, make my day (Clint Eastwood as Dirty Harry)

frankly my dear, I don't give a damn (Clark Gable in Gone With The Wind) *who are those guys* (Paul Newman in Butch Cassidy And The Sundance Kid)

you couldn't handle the truth (Jack Nicholson in A Few Good Men)

this corn is special (Ned Beatty in Deliverance).

See Figure 14.10.

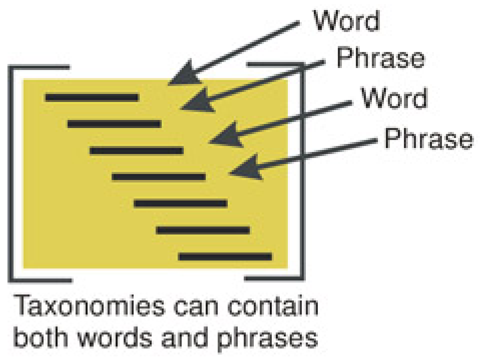


Figure 14.10: Both words and phrases can be found in a taxonomy

Tracing Taxonomies

So where do taxonomies come from? Taxonomies come from categorizations of words found in the real world based on real world relationships. The real world may be the world of business, healthcare, recreation, humor, religion, politics, family history, etc. There simply is no limitation as to where taxonomies can be found.

In some cases, the taxonomies are captured formally, and in other cases, they are captured informally. In some cases, there is a desire to create an exhaustive list of all the words and phrases in a category. In other cases, there is only a need for the creation of a representative list of words found in a category.

Ontologies Explained

There are many different forms of taxonomies. Related to a taxonomy is the ontology. In many regards, an ontology is a superset of a taxonomy. An ontology is a taxonomy in which there are relationships between the different words found in the ontology. See Figure 14.11.

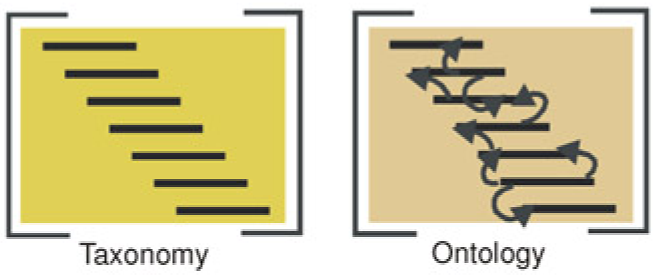


Figure 14.11: A taxonomy and an ontology

One of the interesting features of a taxonomy is that the taxonomy can cross languages. For example, a taxonomy in English can be translated to the same taxonomy in Spanish and lose none of the meaning of the taxonomy. While this may appear as a novel feature of taxonomies, it oftentimes has very real application. For example, suppose that an organization has customers who speak Spanish and other customers who speak English. The text can be collected in both languages. The Spanish version of the taxonomy can then be used to categorize the words in Spanish. The words in Spanish can be translated using the parallel categorization in English. In such a manner, a single language categorization can be created where the raw text is found in more than one language.

Exercise 14: Looking for a Taxonomy

1. Identify at least one taxonomy in use within your organization. Would you consider this taxonomy to be well-defined and fully leveraged in your organization? Why or why not?

Key Points

- There are two basic forms of data — structured data and unstructured data. In order to create an abstraction of structured data, data models are created. But textual data differs fundamentally from structured data in that unstructured data cannot be changed, while structured data can usually be changed.
- In order to create an abstraction of unstructured data, taxonomies are used. The taxonomies are useful for organizing unstructured data into common categories.
- Taxonomies, then, are to unstructured data what data models are to structured data.
- Taxonomies are the basis under which text is abstracted. The abstraction of text is valuable because of the ability to overcome different terminology in text, to use query languages to find classes of data, and to organize text according to the general context of the unstructured data.
- An ontology is a taxonomy in which there are relationships between the different words found in the ontology.

Chapter 15: What is UML?

By Michael Blaha

Overview

Capture the business
Data, process — tightly linked
Enter UML

For the past 15 years, Michael Blaha has been a consultant and trainer in conceiving, architecting, modeling, and designing databases. He also has extensive experience in database reverse engineering for product assessment and business due diligence. He has authored six US patents, five books, and many papers. Blaha received his D.Sc. degree from Washington University in St. Louis and is an alumnus of the GE Global Research Center in Schenectady, NY. His latest book is *Patterns of Data Modeling*. In this chapter, Michael discusses the Unified Modeling Language (UML) from the perspective of database applications. He will cover the diagrams of the UML that are most pertinent to database applications. Note that in this chapter, the term 'model' refers to UML models, not data models.

UML Explained

A UML *model* is an abstraction of an application that lets you thoroughly understand it. Models are used for programming, creating databases, and other purposes—although we emphasize databases here. A model provides a roadmap for an application, similar to a blueprint for a building that is studied and revised many times before it is built. There are many reasons for building software via models:

- **Better quality.** Your application can be no better than the underlying thought. ACM Turing award winner Fred Brooks contends "that conceptual integrity is the most important consideration in system design." [Brooks-1995]
- **Reduced cost.** You can shift your activities towards the relatively inexpensive front end of software development and away from costly debugging and maintenance.
- **Faster time to market.** It is faster to deal with difficulties at the conceptual stage than to deal with them when software has been cast into programming and database code.
- **Better performance.** A sound model simplifies database tuning.
- **Improved communication.** Models reduce misunderstandings and promote consensus between developers, customers, and other stakeholders.

Models are helpful for software that is purchased, as well as software that is developed. You need to understand your requirements and their manifestation in software before you can assess the strengths and weaknesses of various vendor products.

The UML (Unified Modeling Language) is a graphical language for modeling software development artifacts. It spans the range of the lifecycle, from conceptualization to analysis, then design, and ultimately, implementation.

The UML arose from the many object-oriented (OO) approaches to software development that were prevalent in the 1990s. OO approaches had become popular, but the incompatible notations and terminologies were fracturing the software community and causing confusion. In reality, there was little substantive difference between the various approaches. The purpose of the UML was to rise above this babble and standardize concepts and notation so that developers could read each other's models and build on each other's efforts. One benefit of the UML is that a single notation can address both programming and database concerns and bridge the cultural gap across the two communities. Another more subtle benefit is that operations provide a hook for attaching database functionality — stored procedures, referential integrity, triggers, and views.

The UML has been developed under the auspices of the OMG (Object Management Group) [www.omg.org]. The initial formulation was driven by Grady Booch, James Rumbaugh, and Ivar Jacobson, but over the years, there have been dozens of significant contributors. The current release is UML 2.0.

Note that the UML standardizes concepts and notation. The UML does not address the issue of software development process. There are a number of processes in the literature for how to use UML notation to develop software.

The UML has a variety of diagrams, including the following:

- **Class diagram.** Involves classes, relationships, and generalizations. Specifies data structure.
- **Object diagram.** Concerns individual objects and links among objects. Shows examples of data structure.
- **Use case diagram.** Specifies the high-level functionality of software from the perspective of an end-user. Also notes the external actors involved with the software.
- **State diagram.** Concerns states and events that cause transitions between states. Describes the discrete, temporal behavior of objects.
- **Activity diagram.** Shows the workflow for an individual piece of functionality.
- **Sequence diagram.** Shows how processes interact, with whom and what, and in what order.

The UML has been warmly embraced by programmers and is often used during the development of programming code.

However, the UML has had a mixed reception by the database community. Many database practitioners are aware of the UML, but do not use it. The problem is that the UML standardization process ignored the database community. UML jargon emphasizes programming, which puts off many database developers. The irony is that the programming jargon is superficial and in reality, the UML has much to offer for database application development, as this chapter will show.

A strength of the UML is its large variety of diagrams. For general software development, it is likely that the UML will have all the diagrams that you need. For specialized problems, the UML may be lacking and you may need to include additional kinds of diagrams.

A weakness of the UML is its large variety of diagrams. However, do keep in mind that you don't have to use all of the UML models. Just use the ones that are helpful for your applications. We often construct class models. On occasion, we also prepare use cases, state diagrams, activity diagrams, and sequence diagrams.

Modeling Inputs

It is important to consider all inputs when modeling software. In particular, you should be expansive in your search for information and not obsess with use cases. Use cases are significant, but are only one information source. A skilled developer should be adaptable and able to glean requirements from all available resources. Sources of requirements include the following:

- **Use cases.** Many business persons think naturally in terms of use cases and find them convenient for specifying requirements.
- **Business documentation.** Business justification, screen mock-ups, sample reports, and other documentation are often already prepared and available for the asking.
- **User interviews.** Developers can question business experts for missing information and clarification.
- **Technical reviews.** Technologists and line managers can also contribute to the content of a model. For example, they may have prior application or business experience.
- **Related applications.** Consider systems that are to be replaced, as well as systems that will remain, but overlap the new system.

Standard models. Some types of applications have models that are available from standards organizations. For example, the OMG has published the *common warehouse metamodel* to standardize data exchange for data warehouses. [Poole-2002]

Modeling Outputs

Modeling outputs include diagrams, of course, but diagrams alone are insufficient. When we prepare models for applications, we often deliver the following outputs:

- **Diagrams.** Diagrams apply rigor to requirements so that they can be thoroughly considered and acted upon.
- **Explanation.** Sometimes we write a narrative that interleaves diagrams with explanation. On other occasions, especially if our model is delivered via the files of a modeling tool, we prepare a data dictionary and attach definitions to major model elements.
- **Database structure.** You can convert a model into a database schema by applying straightforward rules as [Blaha-2005] explains.

- **Converted data.** Often, there is data from a prior application or related applications that can seed the database for a new application. For example, you may have a customer list to seed a new marketing application.

Class Model Explained

The UML *class model* describes data structure — the classes that are involved and how they relate to one another. The major concepts in the class model are classes, associations, and generalizations. Figure 15.1 shows a sample class model. We will first explain the meaning of the model, and then explain the UML constructs. You'll notice that the class model looks a lot like a data model.

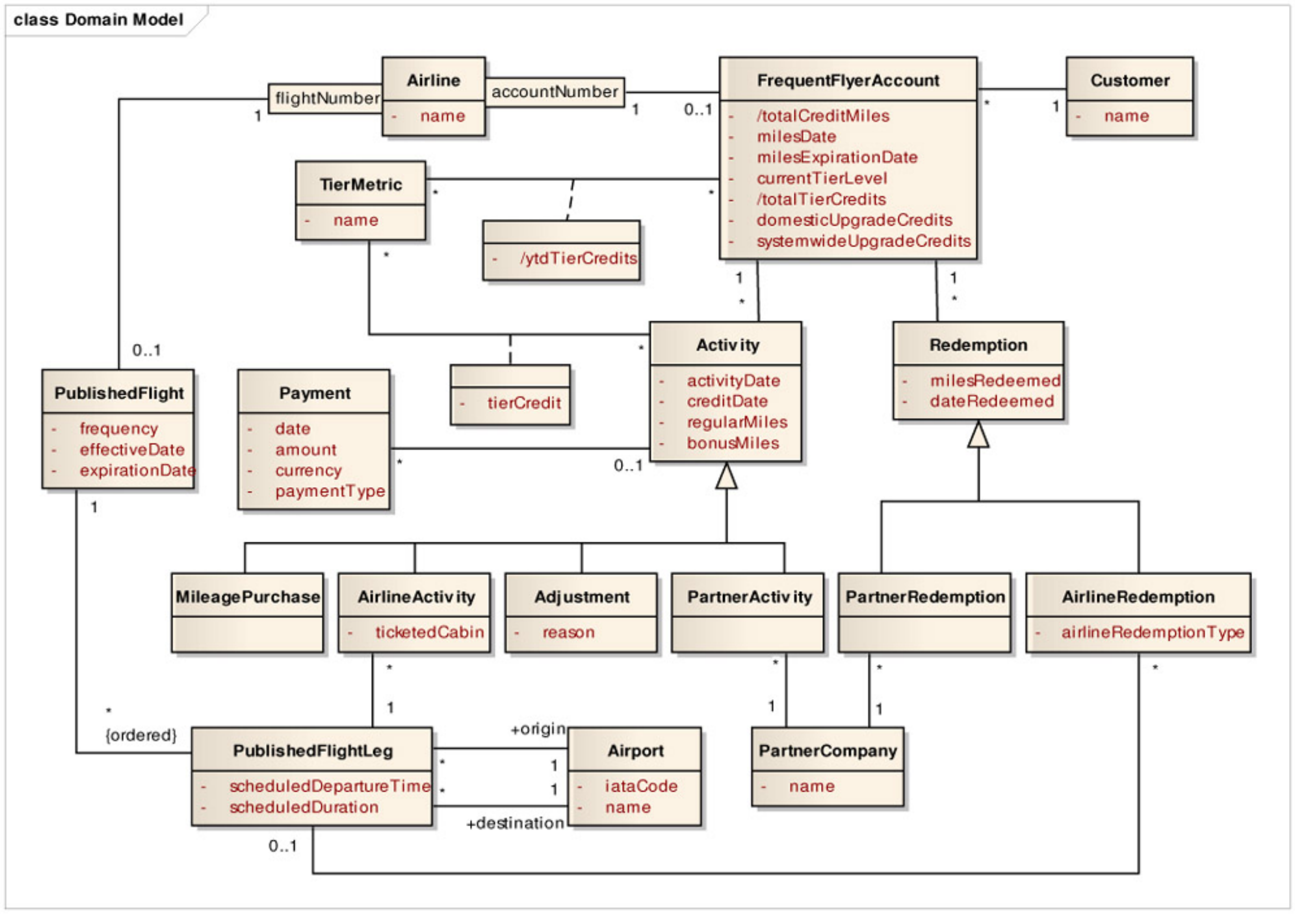


Figure 15.1: Sample class model

In Figure 15.1, a customer can have multiple frequent flyer accounts, though normally a customer has only a single account for an airline. Each frequent flyer account has a unique account number for a given issuing airline.

Each account has a total number of credit miles that can be computed from the underlying activities and redemptions. There is a date as of which the total credit miles are computed, as well as a date that the miles expire. Airline practice is to extend the expiration date as long as there are sufficient activities and redemptions. An account may have upgrade credits to first class for domestic travel. An account may also have system-wide upgrade credits for traveling overseas. A frequent flyer account is a holder for activities that earn miles as well as redemptions that consume miles.

There are various kinds of activities. A mileage purchase is the procurement of credit miles in exchange for payment. Airline activity is the nominal purpose of frequent flyer accounts — in return for airline travel, a customer receives mileage credits. The credits vary according to the ticketed cabin as well as promotional rules that vary over time. The logic for awarding credit is outside the scope of the model. An adjustment corrects an error or posts credit due to unusual circumstances (such as compensation to a customer after a troublesome flight). Partner activity is also important for frequent flyer accounts. Airlines market the accounts to their partners — such as car rental companies, hotels, and credit card companies — who attract customers by paying the airlines for mileage credit.

Similarly, there are various kinds of redemptions. Many redemptions are for free airline flights or to upgrade the cabin from the ticketed fare. Other redemptions are via partners, such as free car rentals and free merchandise.

Airlines offer published flights that have a frequency (such as every day except Saturday). Published flights have effective and expiration dates. A published flight consists of flight legs; each flight leg has an origin airport and a destination airport. The distinction between flights and flight legs is important when dealing with flights. It is airline practice to have some flight numbers that start from a city, stop in an intermediate city, and then arrive in a destination city. The whole sequence has the same flight number and customers can purchase the full flight or any of the constituent legs.

Each account has a current tier level. For example, the tier levels for American Airlines are gold, platinum, and executive platinum. Customers with higher tier levels receive more perks and flying benefits. The accumulated tier credits from account activity determines the customer's tier level. There can be different ways of computing the tier level (tier metric) — American Airlines uses points, miles, and segments (count of traveled flight legs).

Class

An *object* is a concept, abstraction, or thing with identity that has meaning for an application. A *class* describes a group of objects with the same attributes, operations, kinds of relationships, and semantic intent. The UML symbol for a class is a box with the name of the class in the top portion of the box. In Figure 15.1, Airline, FrequentFlyerAccount, and Activity are examples of classes. A class is similar to an entity in a data model, but it is broader in that it includes operations.

The second portion of a class box shows the attribute names for the class. An *attribute* is a named property of a class that describes a value held by each object of the class. In Figure 15.1, FrequentFlyerAccount has seven attributes, Airline has one attribute, and MileagePurchase has zero attributes.

Several of the attribute names in Figure 15.1 are prefaced with a slash (/) symbol. The slash is UML notation for derived data. For example, **totalCreditMiles** is computed (derived) from the posted activities and redemptions.

Although Figure 15.1 does not show it, each attribute can have a multiplicity that specifies the number of possible values for instantiation. The most common options are a mandatory single value [1], an optional single value [0..1], and many [*]. Multiplicity specifies whether an attribute is mandatory or optional (whether an attribute can be null). Multiplicity also indicates if an attribute is single valued or can be a collection of values.

The third portion of a class box (not shown in Figure 15.1) shows the operations for a class. An *operation* is a function or procedure that may be applied to or by objects in a class. Operations are used frequently when using UML class diagrams for programming. Sometimes they are also used for database design, such as when an operation is written as a database stored procedure. For example, a stored procedure could update the **totalCreditMiles** for a FrequentFlyerAccount each time there is an Activity or Redemption. There might be another stored procedure that checks a FrequentFlyerAccount for sufficient credit miles to cover a Redemption.

Association

A *link* is a physical or conceptual connection among objects. An *association* is a description of a group of links with common structure and semantics. The links of an association connect objects from the same classes. An association describes a set of potential links in the same way that a class describes a set of potential objects. The UML notation for an association is a line (possibly with multiple line segments) between classes. In Figure 15.1 the line between PublishedFlight and PublishedFlightLeg is an association. Similarly, there are two lines and two associations between PublishedFlightLeg and Airport. An association is similar to a relationship on a data model.

A binary association has two ends. (The UML supports ternary and higher order associations, even though they seldom occur.) Each end can have a name and multiplicity. *Multiplicity* specifies the number of instances of one class that may relate to a single instance of an associated class. The UML specifies multiplicity with an interval. The most common multiplicities are "1", "0..1", and "*" (the special symbol for "many"). In Figure 15.1, there is one origin Airport and one destination Airport for a PublishedFlightLeg. (Origin and destination are association end names.) An Airport can have many PublishedFlightLegs having it as an origin, as well as many PublishedFlightLegs having it as a destination.

Usually the objects on a "many" association end have no explicit order, and you can regard them as a set. Sometimes, however, the objects do have an explicit order. You can indicate an ordered set of objects by writing "{ordered}" next to the appropriate association end. In Figure 15.1, the

PublishedFlightLegs for a PublishedFlight are ordered. For example, a through flight could first go from St. Louis to Chicago and then from Chicago to Buffalo.

An *association class* is an association that is also a class. Like the links of an association, the instances of an association class derive identity from instances of the component classes. Like a class, an association class can have attributes and operations and participate in associations. The UML notation for an association class is a box that is connected to the corresponding association with a dotted line. Figure 15.1 has two association classes, one between FrequentFlyerAccount and TierMetric and the other between TierMetric and Activity. These two association classes each have one attribute. An association class is similar to an associative entity that resolves a many-to-many relationship.

A *qualified association* is an association in which an attribute called the *qualifier* partially or fully disambiguates the objects for a "many" association end. The qualifier selects among the target objects, reducing the effective multiplicity, often from "many" to "one". Names are often qualifiers. The notation for a qualifier is a small box on the end of the association line near the source class. The source class plus the qualifier yields the target class. Figure 15.1 has two qualified associations. The **accountNumber** for a FrequentFlyerAccount is unique within the context of the issuing Airline. Similarly, the **flightNumber** for a PublishedFlight is unique within the context of an Airline. In data modeling terms, the Airline primary key + **accountNumber** is a unique key for FrequentFlyerAccount, and the Airline primary key + **flightNumber** is a unique key for PublishedFlight.

Aggregation is a strong form of association in which an aggregate object is made of component parts. The most significant property of aggregation is transitivity (if *A* is part of *B* and *B* is part of *C*, then *A* is part of *C*) and anti symmetry (if *A* is part of *B*, then *B* is not part of *A*). Figure 15.1 does not show any aggregations.

Composition is a form of aggregation with two additional constraints. A constituent part can belong to at most one assembly. Figure 15.1 does not show composition.

Generalization

Generalization is the relationship between a class (the *superclass*) and one or more variations of the class (the *subclasses*). Generalization organizes classes by their similarities and differences, structuring the description of objects. The superclass holds common attributes, operations, and associations; the subclasses add specific attributes, operations, and associations. Each subclass *inherits* the attributes, operations, and associations of its superclass. A hollow arrowhead denotes generalization and points to the superclass. You should be familiar with similar concepts in data modeling — subtyping, supertypes, and subtypes.

Figure 15.1 has two generalizations. One generalization has a superclass of Activity and subclasses of MileagePurchase, AirlineActivity, Adjustment, and PartnerActivity. The other generalization has a superclass of Redemption and subclasses of PartnerRedemption and AirlineRedemption.

Use Case Model Explained

The UML *use case model* describes functionality in terms of actors and real-world behavior (use cases) that they perform. The focus is on how the software interacts with outside actors.

Actor

An *actor* is a direct external user of a system. Actors include humans, external devices, and other software systems. A class can be bound to multiple actors if it has different facets to its behavior.

The UML symbol for an actor is a stick figure with the name of the actor as a legend below the figure. Figure 15.2 has several examples of actors and also shows actor generalization (an Agent may be a ComputerAgent or a HumanAgent). Figure 15.3 shows three of the actors from Figure 15.2.

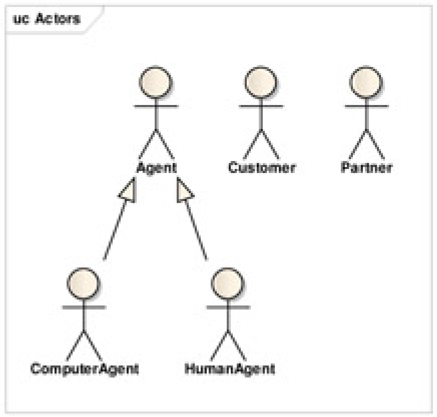


Figure 15.2: Actors for the frequent flyer example

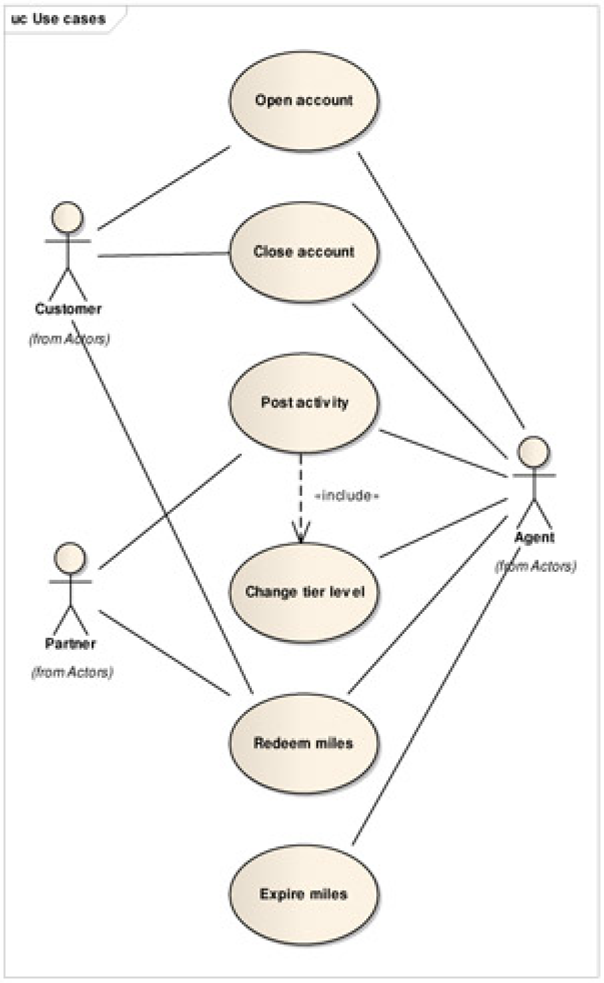


Figure 15.3: Sample use cases

Use Case

A *use case* is a coherent piece of functionality that a system can provide by interacting with actors. Use cases emphasize externally-perceived functionality rather than the details of implementation. The UML symbol for a use case is an oval with the name of the use case inside. A use case is connected to pertinent actors via solid lines.

Figure 15.2 shows actors for the frequent flyer example.

- **Agent.** Agents participate in use cases such as opening and closing an account. We distinguish between human agents and computer agents because they may differ in their precise functionality. The distinction could be important to an airline's business.
- **Customer.** Customer is an obvious actor. The entire purpose of the frequent flyer software is to respond to customers.
- **Partner.** Partner companies are also external entities that interact with the frequent flyer software.

We considered making airline an actor, but it seems that airline functionality occurs via agents that are affiliated with an airline. We should revisit that decision as we add use cases. There may be additional actors that Figure 15.2 does not show. You can find actors by thinking through usage scenarios (use cases) and paying attention to high-level external entities that interact with the software.

Figure 15.3 shows six use cases for the frequent flyer example.

- **Open account.** Create a new frequent flyer account for a customer. Try to ensure that the customer does not already have a frequent flyer account. Set the total credit miles and total tier credits to zero. Post an adjustment if there is a promotional reward for opening a new account.
- **Close account.** Set a date that flags a frequent flyer account as no longer active. (Need to add this to the class model.) After some further passage of time, destroy the frequent flyer account and any associated activity and redemption. This two-phase approach could be helpful if there is a mistake or a change of heart by a customer.
- **Post activity.** Process new activity for a frequent flyer account and add it to the database. Update the total credit miles, miles date, mileage expiration date, and the total tier credits. If there are sufficient tier credits via one of the metrics, invoke the change tier level use case. (Or alternatively, scan all the accounts and change the tier level as appropriate.)
- **Change tier level.** Raise the tier level upon sufficient activity according to a tier metric. Lower the tier level when there has been insufficient activity to maintain a tier level for the past reporting period (a calendar year at American Airlines).
- **Redeem miles.** Give the customer the requested reward and debit his/her frequent flyer account. Ensure that the account has sufficient miles to cover the requested reward.
- **Expire miles.** Post an adjustment to zero the total credit miles if the account has not had activity within a specified time interval. (The time interval is a business policy. The current policy for American Airlines is that an account must have activity every 18 months or the total is zeroed.)

Note that the use cases have different combinations of actors. All the use cases interact with *Agent*. Only two use cases interact with *Partner* and three interact with *Customer*.

The *Post activity* use case includes the *Change tier level* use case, as the posting of an activity can cause a change of tier level. The *includes* is an example of a use case relationship and is shown with an arrow and an annotated dotted line.

Exercise 15: Creating a Use Case

1. Find an opportunity to build a use case. Did you find the use case to be essential in developing the requirements deliverable?

Key Points

- A UML model is an abstraction of an application that lets you thoroughly understand it. The most significant UML models for databases are the class model and the use case model.
- The UML class model describes data structure — the classes that are involved and how they relate to one another. The major concepts in the class model are classes, associations, and generalizations.
- An object is a concept, abstraction, or thing with identity that has meaning for an application. A class describes a group of objects with the same attributes, operations, kinds of relationships, and semantic intent. An attribute is a named property of a class that describes a value held by each object of the class. An operation is a function or procedure that may be applied to or by objects in a class.
- A link is a physical or conceptual connection among objects. An association is a description of a group of links with common structure and semantics.
- The UML use case model describes functionality in terms of actors and real-world behavior (use cases) that they perform. An actor is a direct external user of a system.

References

[Blaha-2005] Michael Blaha and James Rumbaugh. *Object-Oriented Modeling and Design with UML, 2nd Edition*. Upper Saddle River, NJ: Prentice Hall, 2005.

[Brooks-1995] Frederick P. Brooks, Jr. *The Mythical Man-Month, Anniversary Edition*. Boston, MA: Addison-Wesley, 1995.

[Chen-1976] PPS Chen. The Entity-Relationship model—Toward a Unified View of Data. *ACM Transactions on Database Systems 1*, 1 (March 1976).

[Poole-2002] John Poole, Dan Chang, Douglas Tolbert, and David Mellor. *Common Warehouse Metamodel*. New York: OMG Press, 2002.

[Rumbaugh-2005] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual, Second Edition*. Boston, MA: Addison-Wesley, 2005.

Chapter 16: What are the Top Five Most Frequently Asked Modeling Questions?

Overview

Many industries
Yet similar challenges
Learn once, use often

This chapter contains the top five questions that have been asked over the last few years during data modeling training. If you have a question that is not listed here, feel free to email it to me at me@stevehoberman.com.

1. What is Metadata?

When most IT people are asked for their definition of metadata, before we can even think about what it really means, we hear ourselves say out loud, "Data about data." This is, however, a poor definition. As we know from our discussion of definitions in Chapter 12 on the Data Model Scorecard, a good definition is clear, complete, and correct. Although the "Data about data" definition is correct, it is not clear or complete. It is not clear, since businesspeople need to understand what metadata means and I have never found someone who responds with "Ah, I get it now" after hearing the "data about data" explanation. It is not a complete definition because it does not provide examples nor distinguish how metadata really differs from data. So we need a better definition for the term metadata.

I spoke on a metadata-related topic with several DAMA chapters and user groups. Collectively, we came up with the following definition for metadata:

- *Metadata is text, voice, or image that describes what the audience wants or needs to see or experience. The audience could be a person, group, or software program. Metadata is important because it aids in clarifying and finding the actual data.*
- *A particular context or usage can turn what we traditionally consider data into metadata. For example, search engines allow users to enter keywords to retrieve web pages. These keywords are traditionally data, but in the context of search engines they play the role of metadata. Much the same way that a particular person can be an Employee in one role and a Customer in another role, text, voice or image can play different roles - sometimes playing 'data' and sometimes playing 'metadata', depending on what is important to a particular subject or activity.*
- *There are at least six types of metadata: business (also known as 'semantic'), storage, process, display, project, and program.*
- *Examples of business metadata are definitions, tags, and business names. Examples of storage metadata are database column names, formats, and volumetrics. Examples of process metadata are source/target mappings, data load metrics, and transformation logic. Examples of display metadata are display format, screen colors, and screen type (i.e. IPod vs laptop). Examples of project metadata are functional requirements, project plans, and weekly status reports. Examples of program metadata are the Zachman Framework, DAMA-DMBOK, and naming standards documentation.*
- ## 2. How Do You Quantify the Value of the Logical Data Model?
- The logical data model captures how the business works independent of technology. This view must be understood and agreed on before any development can take place. Skipping the logical data model leads to assumptions on how the business works and what they need from an application. When a new house is built, the architect uses a blueprint as a communication medium with the homeowners. The blueprint is the business solution to

the home, in the same way as the data model is the business solution to an application. If an architect decides to build a house without a blueprint, many assumptions will be made during construction, and there is a very good chance the customer (homeowner) will not be satisfied with the results.

- It can be challenging to put a financial value or other quantifiable measure on the value of a particular logical data model, for the same reason it can be challenging to put a financial value on a particular blueprint. How much is a robust database design or accurate blueprint worth? Instead of quantifying the value of a logical data model, we can quantify the *cost* of not having a logical data model through symptoms such as poor data quality. Find stories on the internet or front page of the newspaper illustrating what happens when there is poor information available. Get good at telling these stories and make sure to include the figures on how much these companies lost because of a data quality issue, or how much could have been saved in terms of money or credibility if the data was of high quality.
- For example, Ben Ettlinger, a lead data administrator, is a friend of mine who illustrates the importance of data quality, and therefore the data model, with a story about how NASA lost a $125 million Mars orbiter because one engineering team used metric units while another used English units for a key spacecraft operation. The logical data model can eliminate or minimize data quality issues such as this.

3. Where Does XML Fit?

Extensible Markup Language (XML) is a type of data model which displays information in a hierarchy format using human-readable tags, allowing both people and software applications to more easily exchange and share information. XML is both useful and powerful for the same reasons any data model is useful and powerful: it is easy to understand, can be technology-independent, and enables representing complex problems with simple syntax. Similar to distinguishing subject area models from logical data models from physical data models, XML distinguishes the data content from formatting (e.g. blue, Arial, 15 point font) from rules. XML rules are represented through a schema, such as a Document Type Definition (DTD) or XML Schema Document (XSD). The schema specifies the rules for the data in an XML document in much the same way as a data model specifies the rules for the data in a database structure. The XML data content is displayed in the form of an XML document and is equivalent to one or more entity instances on a data model.

Figure 16.1 contains an XML document based on an example from Wikipedia.

```
<recipe name="bread" prep_time="5 mins" cook_time="3 hours">

  <title>Basic bread</title>

  <ingredient amount="8" unit="cup">Flour</ingredient>

  <ingredient amount="10" unit="grams">Yeast</ingredient>
  <ingredient amount="4" unit="cup"
       state="warm">Water</ingredient>

  <ingredient amount="1" unit="teaspoon">Salt</ingredient>

  <instructions>

   <step>Mix all ingredients together.</step>

   <step>Knead thoroughly.</step>

   <step>Cover with a cloth, and leave for one hour.</step>

   <step>Knead again.</step>

   <step>Place in a bread baking tin.</step>

   <step>Cover with a cloth, and leave for one hour.</step>

   <step>Bake at 180 degrees Celsius for 30 minutes.</step>

  </instructions>

</recipe>
```

Figure 16.1: Recipe XML document

The terms such as '<step>' are called 'tags'. The data within each pair of tags is called a 'value'. So the value for the tag pair <title> and </title> is 'Basic bread'.

Along with the schema for this XML document, we can learn quite a bit about the actual data. For example, we know that a Recipe can contain Ingredients and Instructions can contain Steps. Because XML is hierarchy-based, however, the rules only go one way. That is, we know a Recipe can contain Ingredients, but can an Ingredient belong to more than one Recipe? I email out monthly Design Challenges (add your email address at www.stevehoberman.com), and in a recent Design Challenge Norman Daoust, business analysis consultant and trainer, summarized this: "XML documents frequently only indicate the cardinality of relationships on one end of the relationship, not both ends."

Therefore, we can take this XML document and schema, and by asking additional business questions, derive a logical data model such as that in Figure 16.2. Sample values are shown in Table 16.1.

Table 16.1: Recipe logical data model values Open table as spreadsheet

Recipe

Recipe Id	Recipe Short Name	Recipe Long Name	Recipe Preparation Time	Recipe Preparation Time Unit Of Measure Code	Recipe Cook Time	Recipe Cook Time Unit Of Measure Code
123	bread	Basic bread	5	01	3	02

➡Open table as spreadsheet

Unit Of Measure	
Unit Of Measure Code	**Unit Of Measure Name**
01	Minute
02	Hour
03	Cup
04	Gram
05	Teaspoon

➡Open table as spreadsheet

Ingredient	
Ingredient Id	**Ingredient Name**
1	Flour
2	Yeast
3	Water
4	Salt

➡Open table as spreadsheet

Recipe Ingredient			
Recipe Id	**Ingredient Id**	**Unit Of Measure Code**	**Ingredient Amount**
123	1	03	8
123	2	04	10
123	3	03	4
123	4	05	1

➡Open table as spreadsheet

Recipe Step			
Recipe Step Id	**Recipe Id**	**Recipe Step Sequence Number**	**Recipe Step Instruction Text**
45	123	1	Mix all ingredients together.
46	123	2	Knead thoroughly.
47	123	3	Cover with a cloth, and leave for one

Recipe Step			
Recipe Step Id	**Recipe Id**	**Recipe Step Sequence Number**	**Recipe Step Instruction Text**
			hour.
48	123	4	Knead again.
49	123	5	Place in a bread baking tin.
50	123	6	Cover with a cloth, and leave for one hour.
51	123	7	Bake at 180 degrees Celsius for 30 minutes.

➡Open table as spreadsheet

Recipe Ingredient Step			
Recipe Step Id	**Recipe Id**	**Ingredient Id**	**Recipe Ingredient Step Instruction Text**
45	123	1	
45	123	2	
45	123	3	
45	123	4	

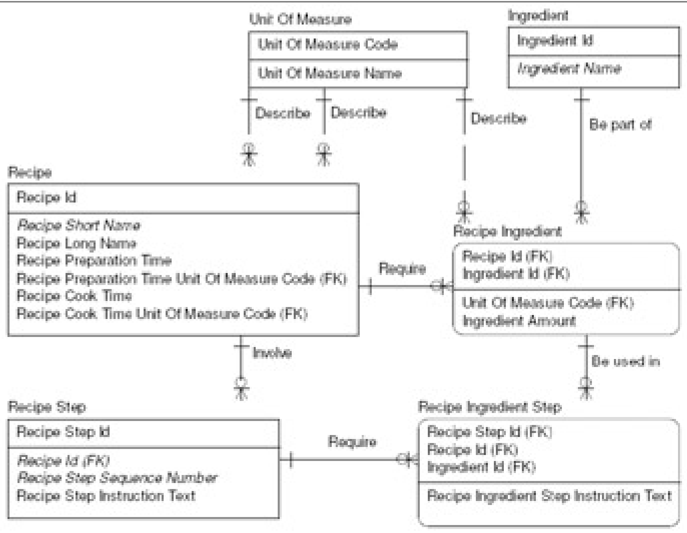


Figure 16.2: Recipe logical data model

Take note all of the business questions that would need to be added to arrive at this model, a small sample being:

- Can the recipe have more than one long name (i.e. title)?

- Is Recipe Short Name really the natural key for Recipe?
- Is Ingredient Name really the natural key for Ingredient?
- Can an Ingredient belong to more than one Recipe?
- Can a Recipe Step require more than one Recipe Ingredient?

XML is in widespread use in our industry. The analyst and modeler can leverage XML to better understand a business area and build a more accurate data model. This is especially true in modeling industry standards. There are many industries that formalized on XML standards so that they can exchange information. One example is ePub, an XML standard for exchanging publishing information across organizations within the publishing industry. Leveraging these standards can make it possible to build more accurate enterprise data models and more useful applications that use industry-wide terminology and rules.

4. Where Does Agile Fit?

Agile means 'quick' and 'adept'. When applied to application development, agile translates into rapid delivery of high-quality software. Agile usually means there are many project-focused iterations until the project is complete. Proponents of agile say the project-focused approach produces high quality results in much less time than a typical software development methodology. Opponents of agile say the focus of agile is on the project at the expense of the program, meaning usually the enterprise perspective and 'big picture' are not given the attention they require. This is not a book on agile pros and cons, so I will stop with this discussion.

There are really two questions here: "Where are data models in an agile environment?" and "Where should data modeling take place in an agile environment?"

The answer to the first question is that usually data models are non-existent or of poor quality in an agile environment. I have taught classes onsite for many organizations using an agile approach and have consistently witnessed few data models been built. The reason is that agile focuses more on process and prototypes than data snapshots and building data models takes valuable time away from software development.

The answer to the second question, "Where should data modeling take place in an agile environment?" is the same as for any other project. Data modeling should be a process to find out what the business wants and to document these requirements consistent with other organization data models and views. Data modeling requires asking lots of questions — regardless of the software methodology chosen, these questions still need to be asked.

5. How Do I Keep My Modeling Skills Sharp?

Look for every opportunity to model or participate in the analysis and modeling process, even outside the traditional roles of a data modeler. The more roles we play around the modeling space, the greater our modeling skills become. For example, after modeling for a number of years, I decided to try development. As a developer, I became one of the customers of the data model, looking at the model with a different eye from that of the modeler. I became able to anticipate questions such as these in my design:

- How can I efficiently populate this structure with an extract, transform, and load (ETL) tool?
- Are there minor modifications I can make to the data model to make the development less complex and take less time?
- How can we extract data out as rapidly as possible for reporting?

Thinking of these questions during my modeling helped me become a more pragmatic data modeler. It broadened my view on the physical data model, and when I returned to modeling, I anticipated many of the questions I knew the developers needed to know on the physical data model.

This may sound geeky, but I also find myself modeling forms and documents that I encounter in my personal life. For example, I might sketch the data model for a menu while waiting to order food in a restaurant. I remember looking at the label of a bottle of prescription medicine and being surprised how concatenated and multi-valued some of the fields that were printed on the label were. I therefore started sketching what the ideal data model to store all this prescription information and correct some of these data problems would be.

There are some excellent books to read and websites that contain newsletters and other valuable information that I've listed under the Suggested Reading section. Also, if you visit my Web site, www.stevehoberman.com, you can add your email address to the design challenge list. Once a month, I email a data modeling puzzle to everyone on the list, then I consolidate everyone's responses and publish them in a well-known industry publication.

There are conferences, courses, and data organizations that keep us in touch with the industry. I teach a full four-day class on data modeling called the Data Modeling Master Class (see www.stevehoberman.com/DataModelingMasterClass.pdf). The Data Warehousing Institute (www.tdwi.org) offers some in-depth courses and conferences on data modeling and business intelligence. eLearningCurve.com offers some great online courses. DAMA (www.dama.org) has a very large annual conference; there are also many local DAMA chapters with monthly meetings.

Books

Adelman S., Moss L., Abai M. 2005. *Data Strategy*. Boston, MA: Addison-Wesley Publishing Company.

Blaha M., Rumbaugh J., 2004. *Object-Oriented Modeling and Design with UML, second edition,* Boston, MA: Prentice Hall.

DAMA International 2009. *Data Management Body of Knowledge (DAMA-DMBOK),* New Jersey: Technics Publications, LLC.

Eckerson, W. 2005. *Performance Dashboards: Measuring, Monitoring, and Managing Your Business*. New York: John Wiley & Sons, Inc.

Hay, D. 1995. *Data Modeling Patterns*. Dorset House Publishing Company, Incorporated.

Hoberman, S. 2001. *The Data Modeler's Workbench*. New York: John Wiley & Sons, Inc.

Hoberman, S., Burbank, D., Bradley C. 2009. *Data Modeling for the Business*, New Jersey: Technics Publications, LLC.

Imhoff C., Galemmo, N., Geiger, J. 2003. *Mastering Data Warehouse Design: Relational and Dimensional Techniques*. New York: John Wiley & Sons, Inc.

Inmon W., Nesavich A. 2008. *Tapping into Unstructured Data*. Boston, MA: Prentice Hall.

Kimball R., Ross M., Thornthwaite W., Mundy J., Becker B. 2008. *The Data Warehouse Lifecycle Toolkit: Practical Techniques for Building Data Warehouse and Business Intelligence Systems*. Second Edition. New York: John Wiley & Sons, Inc.

Marco D., Jennings M. 2004. *Universal Metadata Models*. New York: John Wiley & Sons, Inc.

Maydanchik, A. 2007. *Data Quality Assessment*. New Jersey: Technics Publications, LLC.

Mosley, M. 2005. *DAMA Dictionary of Data Management*. New Jersey: Technics Publications, LLC.

Potts, C. 2008. *fruITion: Creating the Ultimate Corporate Strategy for Information Technology*. New Jersey: Technics Publications, LLC.

Silverston, L. 2001. *The Data Model Resource Book, Revised Edition, Volume 1, A Library of Universal Data Models For All Enterprises*. New York: John Wiley & Sons, Inc.

Silverston, L. 2001. *The Data Model Resource Book, Revised Edition, Volume 2, A Library of Universal Data Models For Industry Types*. New York: John Wiley & Sons, Inc.

Silverston, L. Agnew, P. 2009. *The Data Model Resource Book, Volume 3, Universal Patterns for Data Modeling*. New York: John Wiley & Sons, Inc.

Simsion, G. 2007. *Data Modeling Theory and Practice*. New Jersey: Technics Publications, LLC.

Simsion G., Witt G. 2005. *Data Modeling Essentials, Third Edition*. San Francisco: Morgan Kaufmann Publishers.

Web Sites

www.dama.org — Conferences, chapter information, and articles

www.eLearningCurve.com — offers some great online courses

www.information-management.com — I use the article search mechanism often.

www.metadata-standards.org/11179 — Formulation of data definition

www.stevehoberman.com — Add your email address to the Design Challenge list to receive modeling puzzles

www.tdan.com — In-depth quarterly newsletter. I visit this site very often to read the current newsletter or search the archives.

www.tdwi.org — Great conferences, seminars, and white papers

www.teradata.com — Great white papers, even if your organization does not currently use Teradata

Appendix A: Answers to Exercises

"Answers" is a strong word. It implies I know *the* answer as opposed to knowing *an* answer, which is closer to the truth. In other words, you may have different and possibly better answers than I, and that would be a very a good thing!

Exercise 1: Educating Your Neighbor

1. Reinforce your own understanding of what a data model is by explaining the concept of a data model to someone completely outside the world of IT, such as to a neighbor, family member or friend.

   Did they get it?

Answers

1. I find the analogy that I use most frequently is comparing the data model to a blueprint. Most non-technical friends, family, and neighbors understand this analogy. "Just like you need a blueprint to ensure a sound building structure, you need a data model to ensure a sound application." Sometimes I also explain to people that a data model is nothing more than a fancy spreadsheet, which contains not just the spreadsheet columns, but also the business rules binding these columns. If both the blueprint and spreadsheet analogies fail, I quickly change the subject to the other person and ask what they do (and hope they never ask me again!).

Exercise 3: Choosing the Right Setting

1. In the following table, check off the most appropriate settings for each of these scenarios. See the Appendix for my answers.

1. Explain to a team of developers how an existing contact management application works

   Open table as spreadsheet

Scope	Abstraction	Time	Function
❑ Dept	❑ Bus clouds	❑ Today	❑ Bus
❑ Org	❑ DB clouds	❑ Tomorrow	❑ App
❑ Industry	❑ On the ground		

2. Explain the key manufacturing concepts to a new hire

   Open table as spreadsheet

Scope	Abstraction	Time	Function
❑ Dept	❑ Bus clouds	❑ Today	❑ Bus
❑ Org	❑ DB clouds	❑ Tomorrow	❑ App
❑ Industry	❑ On the ground		

3. Capture the detailed requirements for a new sales datamart (A datamart is a repository of data that is designed to meet the needs of a specific set of users)

   Open table as spreadsheet

Scope	Abstraction	Time	Function
❑ Dept	❑ Bus clouds	❑ Today	❑ Bus
❑ Org	❑ DB clouds	❑ Tomorrow	❑ App
❑ Industry	❑ On the ground		

Answers

1. In the following table, I checked off the most appropriate settings for each of these scenarios.

1. Explain how a contact management legacy application works to a team of developers

   ➡Open table as spreadsheet

Scope	Abstraction	Time	Function
☒Dept	☒Bus clouds	☒Today	❑ Bus
❑ Org	❑ DB clouds	❑ Tomorrow	☒App
❑ Industry	❑ On the ground		

2. Explain the key manufacturing concepts to a new hire

   ➡Open table as spreadsheet

Scope	Abstraction	Time	Function
❑ Dept	❑ Bus clouds	☒Today	☒Bus
☒Org	❑ DB clouds	❑ Tomorrow	❑ App
❑ Industry	☒On the ground		

3. Capture the detailed requirements for a new sales datamart

   ➡Open table as spreadsheet

Scope	Abstraction	Time	Function
☒Dept	❑ Bus clouds	❑ Today	☒Bus
❑ Org	❑ DB clouds	☒Tomorrow	❑ App
❑ Industry	☒On the ground		