CHAPTER 3

Designing a Relational Database

In This Chapter

Stating the problem

Brainstorming

Modeling entities and relationships

Building the database

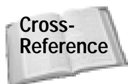
In this chapter, I'm going to show you the way I design a database. While I'll use SQL Server tools in this chapter, the same concepts can be applied to any database system. The key to using this approach is to understand the techniques I'm going to show you and adapt them so that they work for you.

Overview of the Design Process

Designing a relational database can be as easy or as hard as you choose to make it. I generally use a seven-step approach as outlined below:

1. Stating the problem
2. Brainstorming for ideas
3. Modeling entities and relationships
4. Building the database
5. Creating the application

I'm going to discuss the first four steps in this chapter. The rest of this book will focus on step five.



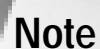
See Chapter 2 for information about the relational database model.

Stating the Problem

While stating the problem may seem easy, it's a lot harder than it looks. The problem statement should present an understanding of what the organization is trying to accomplish, while at the same time trying to emphasize the most critical business needs. If you want to replace an existing application, you can use that application as a basis for the answer. But if you are building an application for the first time, it is very important to understand what problem you're trying to solve. After all, if you can't identify the problem, how are you going to know if you really solved it?

The problem should be stated as simply as possible. For example, if the people at Amazon.com were to state their problem, it might look like this: we want to establish Amazon.com as a brand name by selling books to consumers via the Internet at the lowest possible cost to the consumer and with the best possible service, while building market share that will ensure the long-term stability of the company. A specialty mail-order catalog company might want to solve this problem: we want to improve how we take customer orders over the telephone to reduce mistakes and improve service to our customers. A small electronics supplier might state their problem in this way: we need to improve our inventory control to increase the number of times the inventory is turned over each year and to prevent stockpiles of obsolete items. In this book, I'm going to use a common database for all of the examples I create. I'll also update an application I wrote a few years ago called Car Collector, which tracks a collection of toy cars. Since I originally wrote this program, the Internet has become very important to collectors. Web sites like eBay make it much easier for people to buy and sell collectibles. In fact, my wife likes to buy and sell collectible dolls and has been asking me to build her a version of Car Collector that is targeted at the doll market.

Taking this information into consideration, I can state my problem like this: I want to update Car Collector to include support for other types of toys and collectibles and add support for trading toys with other collectors over the Internet. With the new features I'm going to add, I'm going to change the name of the application to Toy Collector.



Note

It's almost realistic: Toy Collector as used in this book isn't meant to be a complete application, but rather a framework for showing you different techniques for building database applications in Visual Basic. Therefore, the database design may not be as complete as a commercial application, nor will all of the features found in the commercial application be present in Toy Collector. Some of the data structures I use, as well as the features included in the application, may seem like overkill, but they are necessary to illustrate various points along the way. So, focus on the techniques that I'm going to show you and try to understand why and how I do things, rather than focusing on why this feature was added or why that information is missing.

Brainstorming

Once you have a basic understanding of the problem your application needs to solve, you need to design the database to accommodate the information related to your application. The first step in this process it to take a look at your problem and try to determine all of the information and functions that might be needed to solve the problem. I call this step *brainstorming.*

Brainstorming is the act of discussing and recording ideas without regard to how feasible they are to implement. This helps you identify all of the information you need to keep and all of the tasks your application will need to perform.

I like to conduct brainstorming sessions that include everyone who will be involved with the project in a single room with a white board. It helps to have as wide a range of people present as possible. Everyone from end users, to programmers, to management should be involved in this process.

Every idea that is raised should be listed on the white board, even if it's similar to an idea that's already listed. After the meeting, the information should be organized, and similar ideas can be combined as a single item. The ideas should be classified as either a task that the application should perform or a piece of information that will need to be kept.

It's important to understand that some of the ideas that come out of the brainstorming session may not be practical. At this stage of the process, you shouldn't worry about practicality. It is far more important to be complete. Sometimes things that seem impractical at this stage may prove easy to implement later, while other ideas that seem easy to implement at first may not prove to be worth the time and effort.

Also, it's important not to make fun of any ideas, no matter how bad they seem. This is especially true for ideas coming from the less technical attendees. Quite often, their ideas and comments may lead to a better understanding of how the application should work.

Brainstorming Toy Collector

Since you can't actively participate in the brainstorming session, I sat down and held one myself, and came up with the following list of functions that need to be performed by Toy Collector application:

- Track items currently in the collection
- Create reports of items in the collection
- Locate a toy in the inventory

- Keep a mailing list of current and potential customers
- Create a Web page with a list of items currently for sale
- Create a Web page with a list of items wanted
- Create an HTML listing for eBay
- Evaluate the condition of a toy
- Track purchases and sales
- Process an order that sells one or more toys to a customer
- Process an order that purchases a toy from a customer

The functions will be used to maintain a database containing the following data elements:

- Toy name
- Manufacturer
- Description
- Year the toy was built
- Value
- Price paid
- Asking price
- Condition
- Condition questions
- Type of toy
- Order information
- Shipping cost
- Sales tax
- Date ordered
- Date received
- Date shipped
- Credit card number
- Expiration date
- Customer name
- Customer's first name
- Customer's last name
- Customer's middle initial
- Customer's address
- Customer's city
- Customer's state
- Image of the collectible

Reviewing the results

After conducting the brainstorming session, you need to review the information you collected and try to eliminate duplicate information and make sure that the information you have is complete. Quite often while you conduct this review, you'll realize that other related information might be useful and should be added to the list.

Examining the functions to be performed

Looking at the above results, you can see a few common threads. First, you need an inventory system that tracks all of the toys in the collection. This is a fairly common application, along with maintaining a mailing list.

The inventory part of an application usually requires a unique identifier for an item in inventory. This wasn't included as part of the original brainstorming session. In a traditional inventory system, the quantity of the item is also included. However, due to the fact that different toys may have different characteristics based on their condition, I choose to ignore the quantity issue and require that each item in the inventory must have its own unique record.

Processing orders isn't difficult. However, a few more items need to be added to what was already identified above. For instance, in order to compute sales tax, you need to know the sales tax rate. For the purposes of this book, I'm going to assume that the sales tax rate is uniform across a state, even though this isn't necessarily true. You also need to capture the customer's name on the credit card, since it may be different than the name they entered into your database. There should also be an option to ship to an alternate destination, rather than a regular mailing address.

The mailing list is pretty straightforward, except that you should give the customer the option to not receive mail. This is important, because even in today's marketplace, people will complain about unwanted mail. Also, you may want to include some additional comments about the customer that would let the user record problems that they may have had during previous transactions.

The customer's name is actually listed twice: once as simply name, and another time as first, middle, and last names. Which way is best really depends on you. Having a separate last name field makes it easy to search on someone's last name. However, using a single field lets you format a name more naturally, which is important when you have people that have suffixes such as Ph.D., Junior, Senior or III. It also allows someone to enter a title such as Mr., Ms., Dr., etc. with fewer problems. In the long run, it really doesn't matter which method you pick as long as you're consistent throughout your database.

Evaluating the condition of a toy can be a fairly complex process. One method is to assign the toy a point value, assuming that it's in mint condition. Then you would ask a series of questions about specific flaws that are possible in the toy. Depending on the response, points will be deducted from the maximum score. The resulting score can then be mapped onto condition value, which in turn allows you to determine the true value of the toy. Since the questions can vary by the type of toy, additional information is needed to determine the type of toy and the questions that will be used to determine the toy's overall condition.

Mapping the results to data types

The last step of the brainstorming session is to map the data elements onto a series of data types. After reviewing the brainstorming information, I like to assemble a list of the data elements that were derived from the session, along with Visual Basic data type and a short description of the elements, such as those shown in Table 3-1.

Table 3-1
Data Elements

Data Element	*Data Type*	*Description*
CustomerName	String	The name of the customer.
Street	String	The street on which the customer lives.
City	String	The city where the customer lives.
State	String	The state where the customer lives.
Zip	String	The proper ZIP code for the customer's address.
Phone	String	The customer's telephone number.
EMailAddress	String	The customer's e-mail address.
MailingList	Boolean	True if the customer wants to receive periodic notices.
OrderNumber	Long	A number that uniquely identifies the order.
OrderStatus	String	The current status of the order.
DateOrdered	Date	The date the order was placed.
DateShipped	Date	The date the order was mailed.
ShippingCost	Currency	The cost to ship the order.
SalesTax	Currency	The amount of sales tax collected.
CreditCardNumber	String	The credit card number used to purchase a toy.
ExpirationDate	String	The expiration date on the credit card.
InventoryId	Int	A unique identifier for an item in the collection.
ToyName	String	The name of the collectible toy.
Manufacturer	String	The toy's manufacturer.
ToyType	String	The type of toy.
ToyDescription	String	A description of the toy.

Data Element	*Data Type*	*Description*
MintValue	Currency	The value of the toy if it was in mint condition.
Condition	Long	A numeric description of the toy's condition.
Question	String	A question used to evaluate a toy's condition.
TrueValue	Currency	The true value of the toy based on its condition.
DatePurchased	Date	The date the toy was purchased.
Image	Picture	A picture of the toy.

Note

US first: When building an e-commerce application, one of the first things you need to plan for is how to handle international issues. For the most part, the only way these issues would affect your database design is that additional data elements, such as Country and CurrencyType, would need to be included, plus you would need to allow additional space for other fields such as ZIP code. Just because I don't include these fields in Toy Collector isn't a good reason for you not to include them in your application.

Modeling Entities and Relationships

The next step in the process is translating the information from the brainstorming session into a database design.

Entity/relationship modeling

Entity/Relationship modeling (also known as E/R modeling) is a way of describing the *relationship* between *entities*. An entity is a thing that can be uniquely identified, such as a toy, a customer, or an order. Associated with the entity is a set of attributes, which helps to describe the entity. Each customer has a name and an address. Each toy has a name and a manufacturer. An order has an order number and a date ordered. Relationships are formed between two entities, such as customers and orders, where a customer places an order for a toy.

When drawing an E/R model, I use rectangles for entities, ellipses for attributes, and diamonds for relationships. In Figure 3-1, you can see a simple E/R model that has two entities (Customers and Orders), with each entity having two attributes (Customers-Address and Name, Orders-Order Number and Date Ordered) and a single relationship (Customer-Order).

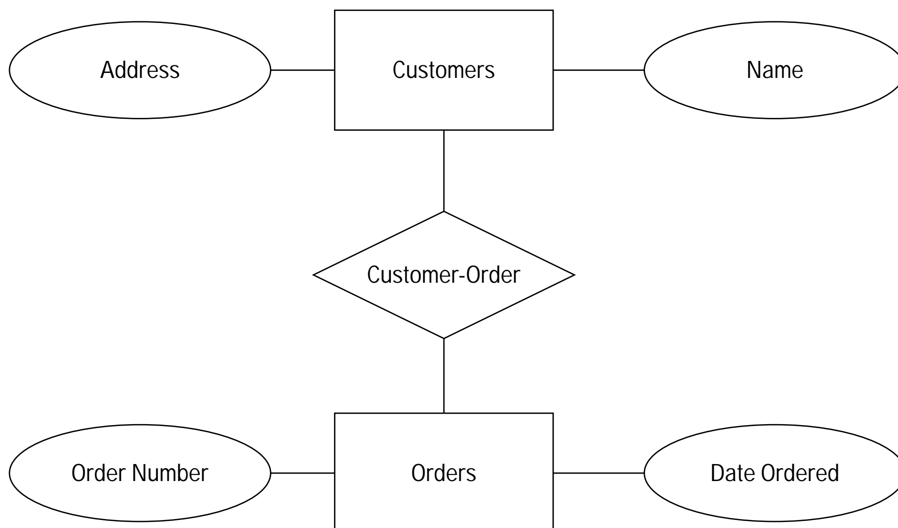


Figure 3-1: Designing a simple database using an E/R model.

Identifying entities and attributes

The first step in this process is to review the list of data elements found in Table 3-1 and look for common groupings. As you scan through the list of data elements, three main groupings jump out almost immediately: customer information, inventory information, and order information. Each of these groupings represents a major entity in the Toy Collector database.

At the same time, you need to look at the various entities and their attributes from an implementation point of view. You may find that a few other attributes are easy to include and will add value to the application from the user's point of view.

Tip

Dirt cheap disk drives: If you designed a database years ago, you will remember that disk space was very expensive, and you always tried to use the least amount of space possible. In today's marketplace, you can purchase a high-performance 9-gigabyte SCSI disk drive for less than $500. If you allow 2,000 bytes for each customer (which is very generous), you can store over 4 million customers on a single disk drive. Since most applications won't store this much data, don't let the cost of disk space drive your database decisions.

Customer information

Table 3-2 contains the list of data elements that are related to a customer, plus a few more that popped up while assembling the list. Finding some additional data elements at this stage is quite normal, since we now have a better understanding of the application's needs. In this case, I added fields to identify when the customer

was originally added to the database (DateAdded) and the last time the information was updated (DateUpdated). I also added a field called Comments that allows the user to record any comments they may have about this particular customer.

Table 3-2
Customer Information

Column Name	*Data Type*	*VB Type*	*Description*
CustomerId	Int	Long	A unique identifier for the customer.
Name	Varchar(64)	String	The customer's name.
Street	Varchar(64)	String	The street address where the customer lives.
City	Varchar(64)	String	The name of the city where the customer lives.
State	Char(2)	String	The name of the state where the customer lives.
Zip	Int	Long	The ZIP code for the customer's address.
Phone	Varchar(32)	String	The customer's phone number.
EmailAddress	Varchar(128)	String	The customer's e-mail address.
DateAdded	Datetime	Date	The date the customer was added to the database.
DateUpdated	Datetime	Date	The date the customer's information was last updated.
MailingList	Bit	Boolean	When true means that the customer wishes to receive periodic mailings.
Comments	Varchar(256)	String	Comments about the customer.

Inventory information

The information about the inventory is a little more complicated than the customer information. While it is easy to identify the attributes that are directly related to an inventory item (see Table 3-3), there are a few cases where some of the information isn't directly related. For instance, the questions that you need to ask the user in order to determine the condition of a toy are related to the type of toy, not the actual toy itself. This implies that another entity called ToyTypes (see Table 3-4) will be needed to hold information about a type of toy. Also, because there are multiple questions for each toy type, you'll need yet another entity (ConditionQuestions) to hold the questions (see Table 3-5).

Table 3-3
Inventory Items

Column Name	*Data Type*	*VB Type*	*Description*
InventoryId	Int	Long	A unique identifier for the item in the collection.
ToyTypeId	Int	Long	A unique identifier for the type of toy in the collection.
Name	Varchar(64)	String	The name of the toy.
ManufacturerId	Int	Long	The name of the manufacturer who made the toy.
YearIssued	Datetime	Date	The date the toy was first manufactured.
Description	Varchar(256)	String	A description of the toy.
MintValue	Money	Currency	The value of the toy if it is in mint condition.
Condition	Int	Long	The condition of the toy using a numeric scale.
ConditionMask	Varchar(64)	String	Answers to the condition questions for this type of toy.
TrueValue	Money	Currency	The true value of the toy based on its current condition.
DatePurchased	Datetime	Date	The date the toy was added to the inventory.
PurchasePrice	Money	Currency	The amount of money paid for the toy.
AskingPrice	Money	Currency	The amount of money you are willing to sell the toy for. A value of zero means that you aren't willing to sell the toy at this time.
BuyingPrice	Money	Currency	The amount of money you are willing to pay for a similar toy.
Wanted	Bit	Boolean	If true means that you want to buy the toy.
ForSale	Bit	Boolean	If true means that you want to sell the toy.

Column Name	Data Type	VB Type	Description
Comments	Varchar(256)	Sting	Any comments that would be displayed along with the toy.
DateUpdated	Datetime	Date	The most recent time this information was updated.

Table 3-4
Toy Types

Column Name	Data Type	VB Type	Description
ToyTypeId	Int	Long	A unique identifier for the type of toy in the collection.
Description	Varchar(64)	String	A description of the type of toy.

Table 3-5
Condition Questions

Column Name	Data Type	VB Type	Description
TypeId	Int	Long	A unique identifier for the type of toy in the collection.
Seq	Int	Long	A sequence number that is used to distinguish between multiple questions for a specific type of toy.
Question	Varchar(64)	String	A question used to evaluate the condition of the toy.
Weight	Int	Long	The relative importance of the question when determining the toy's condition.
Responses	Int	Long	The highest possible value of the response.

Sometimes it is useful to codify a data value to ensure data consistency. A good example of a field that can easily get bad data is the Manufacturer field. Consider how many different ways someone can spell Mattel. One way to ensure that the misspelling doesn't happen is to encode each toy manufacturer as a numeric value and

store the numeric value in the database. Then you need to add a translation table that can be used to translate the codified value into a text string. This is what the Manufacturers table accomplishes (see Table 3-6).

Table 3-6
Manufacturers

Column Name	*Data Type*	*VB Type*	*Description*
ManufacturerId	Int	Long	A unique identifier for the name of the manufacturer.
Name	Varchar(64)	String	The name of the manufacturer.

Another area of concern is that we need to store images for the toys. While I believe it is acceptable for you to store images in a database, I also believe that they should be stored in a separate table. Since I need to use a separate table for the image, I decided to add a sequence number column that will let me store multiple images for a single toy (see Table 3-7).

Table 3-7
Images

Column Name	*Data Type*	*VB Type*	*Description*
InventoryId	Int	Long	A unique identifier for the item in the collection.
Seq	Int	Long	A sequence number that is used to distinguish between multiple images for a single toy.
Image	Image	Picture	A large binary field that holds the actual image of the toy.

Order information

The final major entity that I identified in this application is the Orders entity. However, this entity needs to be subdivided so each order can have multiple items in the order. Thus, Table 3-8 lists the attributes of the Orders entity, while Table 3-9 lists the attributes associated with a single item that is in the order. I called this entity OrderDetails.

Storing Images in a Database

While many people recommend against storing an image in your database, I believe otherwise. By storing images in the database, it is much easier to secure and access them. Storing images outside the database means that you have to maintain a separate security system to protect the images. This can become very complicated if you permit the images to be accessed both by Web browser-based applications and traditional client/server applications.

While I believe in storing images in the database, I also believe that the images should be stored in their own table, away from any related data. Database performance is based mostly on how much information you can retrieve with a single disk I/O. The more rows you can retrieve, the better.

In a typical database table, you might be able to retrieve anywhere from 10 to 100 rows with a single disk I/O. However, if you include an image in the table, you may find that you only get one row for each disk I/O. By moving the image to a separate table, the performance of the main table isn't compromised.

Table 3-8
Orders

Column Name	*Data Type*	*VB Type*	*Description*
OrderId	Int	Long	A unique identifier for the order.
CustomerId	Int	Long	A unique identifier for a customer.
OrderType	Int	Long	1 = sale, 2 = purchase.
ShippingCost	Money	Currency	The total cost of shipping.
SalesTax	Money	Currency	The total cost of sales tax.
HowPaid	Int	Long	1 = credit card, 2 = check.
CreditCardNumber	Varchar(32)	String	The customer's credit card number.
ExpDate	Varchar(16)	String	The customer's credit card expiration date.
OrderStatus	Int	Long	1=order placed, 2=order shipped, 3=order received.
DateOrdered	Datetime	Date	The date and time the order was placed.
DateShipped	Datetime	Date	The date and time the order was shipped.
DateReceived	Datetime	Date	The date and time the order was received.

Table 3-9
OrderDetails

Column Name	*Data Type*	*VB Type*	*Description*
OrderId	Int	Long	A unique identifier for the order.
Seq	Int	Long	A sequence number that is used to distinguish between multiple items in a single order.
InventoryId	Int	Long	A unique identifier for the item in the collection.
PurchasePrice	Money	Currency	The amount paid for the toy.

The last entity I want to talk about is the States entity. This entity exists mostly to translate the two-character state abbreviation into a sales tax rate, which is used to compute the amount of sales tax that must be collected for an order. At the same time, I decided to add the StateName field to translate State into a more meaningful value.

Table 3-10
States

Column Name	*Data Type*	*VB Type*	*Description*
State	Char(2)	String	The two-character abbreviation for a state.
StateName	Varchar(64)	String	The proper state name.
SalesTaxRate	Decimal	Currency	The sales tax rate for the state.

Identifying Relationships

Once you have identified all of your entities and their attributes, identifying the relationships in your design is a piece of cake. You begin by looking at how the entities are related to each other.

There are three basic types of relationships: one-to-one, one-to-many, and many-to-many. These relationships refer to the number instances of data in one entity that are related to instances of data in another entity. In a *one-to-one relationship,* there is only one instance of data in one entity that is related to a single instance of data in another entity. For instance, assume that you have two entities — stores and managers. Each store has a single manager, while each manager has a single store. Thus each store has a unique manager and each manager has a unique store.

In a *one-to-many relationship,* one instance of data in the first entity is related to zero or more instances of data in the second entity. For example, assume that you have an entity for customers and an entity for orders. Each customer may place as many orders as they desire. They need not have placed any orders if they have signed up to be on a mailing list. For each order, there is exactly one customer who placed the order. Thus for each order there is only one customer and for each customer there may be zero or more orders.

In a *many-to-many relationship,* multiple instances of data in the first entity are related to multiple instances of data in the second entity. This can be illustrated by having an entity for parents and an entity for children. Each parent may have zero or more children, while each child may have multiple parents. (Remember, an orphan child has no parents, while a child with divorced parents, may have a mother, a father, a stepmother and a stepfather.)

Drawing the E/R model

Drawing the E/R model is a fairly simple task (see Figure 3-2) with the information found in Tables 3-2 to 3-10. While I didn't list the attributes for each entity because it would render the small drawing nearly unreadable, it is a fairly easy task. Of course, comparing the above tables to the diagram is probably even more meaningful.

When drawing an E/R model, I suggest using a tool like Visio rather than creating a drawing with a paper and pencil. Visio allows you to easily edit the drawing to accommodate the inevitable changes that will occur as various people review and comment on your document. Of course, there are some very expensive database design tools that offer similar capabilities, but I find Visio works nearly as well for most database designs.

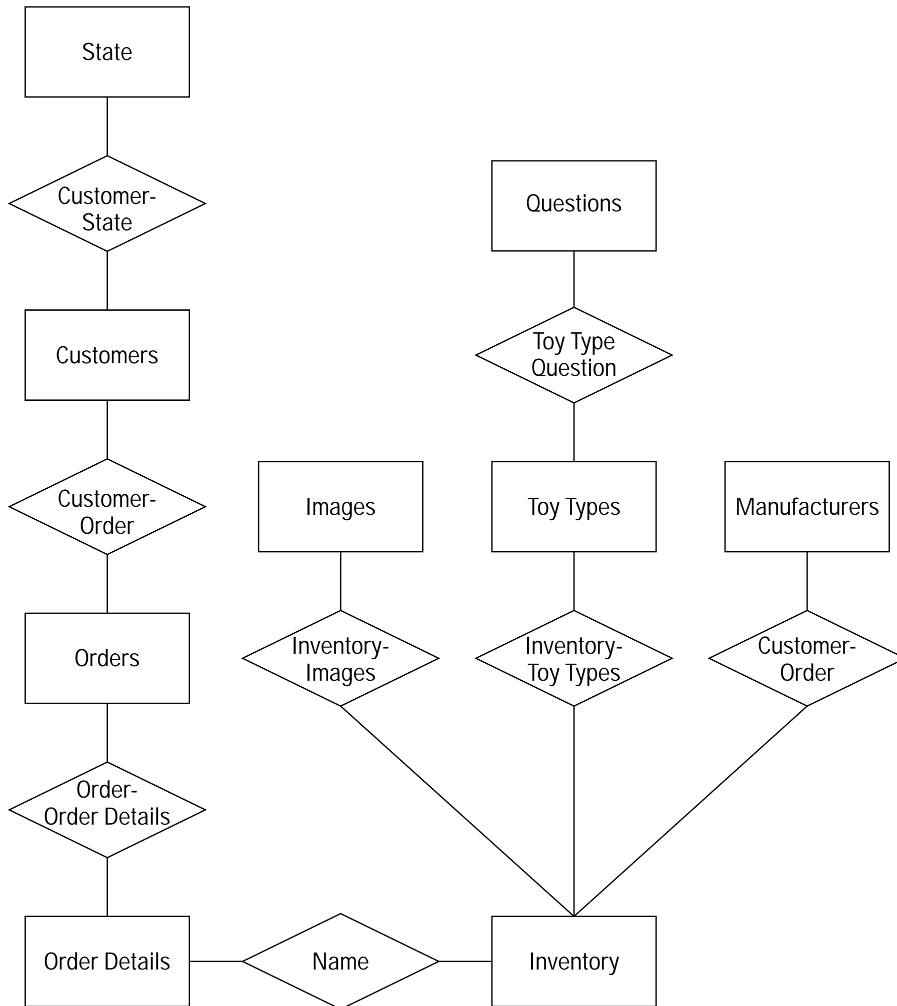


Figure 3-2: Viewing the final Entity/Relationship model for Toy Collector.

Building the Database

Translating an E/R model into a database is a pretty straightforward process. Each of the entities becomes a table and their attributes become columns in the table. You can see the final product in Figure 3-3 using the SQL Server database diagram facility.

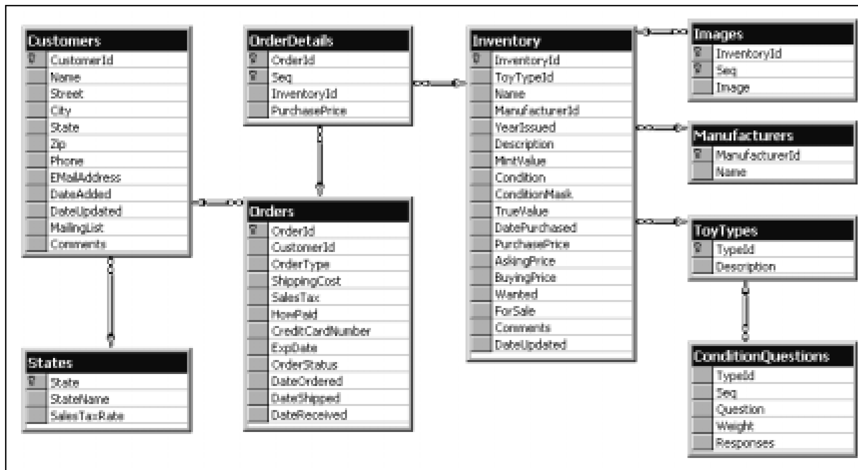


Figure 3-3: Looking at a database diagram of the Toy Collector database.

Thoughts on Database Design

Just because you have a valid database design doesn't mean that you will get the best performance from it. There are a number of factors that will affect performance, such as the number of tables in the database, the size of the columns, and the number of indexes you are using. However, the biggest single factor that affects your database's performance is the hardware you're using.

Believe it or not, having a faster CPU will not necessarily make your database server run faster. A database server is very I/O intensive. Anything that allows the database to retrieve data faster from disk will help the database server's overall performance.

Adding memory to your server allows the database server to cache more data in memory. After all, retrieving data from memory is much faster than retrieving it from disk. This is the biggest change you can make to improve database performance.

After adding memory to your system, using SCSI disk drives in place of IDE drives is the next place you should look for performance gains. Not only can you manage up to 15 disk drives on a single SCSI card, SCSI also allows you to perform concurrent operations on each drive. Thus you can have multiple disk drives performing seeks, while other drives are transferring data. SCSI-III can transfer data faster than SCSI-II or SCSI-I and should be used for best performance.

Using faster disk drives themselves will also improve performance. Disks that spin at 7,200 revolutions per minute (RPM) will transfer data faster than those that spin at 5,400 RPM, although two 5,400 RPM disk drives will probably perform better than one 7,200 RPM drive, assuming that you can split your workload evenly between the two drives. Of course, if you can spring for two of the new 10,000 RPM disk drives, you'll be better off in the long run.

Summary

In this chapter you learned:

- ✦ The five steps in an application design process.
- ✦ Why stating the program helps you clarify goals and objectives for the entire design process.
- ✦ How to use brainstorming to determine the data elements and functions required in your application.
- ✦ How to use Entity/Relationship modeling to design your database.