3. Data Modeling 101

I have an old cardboard box under my desk[4]. When it was delivered a few years ago, it held 10 reams of copy paper. These days it's full of old pay stubs, paid bills, insurance policies, letters from my bank, and other important financial documents waiting to be transferred to their permanent home in my filing cabinet.

For some purposes, tossing the unsorted papers into my box is sufficient. It's a convenient place to temporarily gather everything into a single container. The difficulty comes when I need to find a document from a few weeks ago. I don't know if it's still in the old cardboard box, or if I've already filed it somewhere in my neatly organized filing cabinet. If I'm lucky, it's in the filing cabinet and I can locate it quickly using the filing system I came up with when I first started using it a few years ago. If it's not, I have no choice but to dig through everything in the box, one piece of paper at a time until I find it. That, I can tell you, isn't much fun.

The Cardboard Box Metaphor

By now you've figured out I'm not really talking about a cardboard box and a four-drawer filing cabinet (although they are real enough features of my office). They are metaphors for some of the things we need to understand in order to build a robust, efficient, and above all, useful database.

In its most fundamental sense, my box is a *database.* That is to say, *it is a collection of data about a particular subject or purpose*. In its current state of (non) organization, my "box database" isn't very useful except, of course, as short-term storage.

My four-drawer filing cabinet comes closer to being a real database. It, too, is a collection. However, it is a collection of information on a particular subject or purpose, organized according to a pre-defined structure, or model.

[4] I won't hold it against you if you skip this chapter and run ahead to start building your Access tables. If you are confident you know what a Data Model is and how to build one, and if you can recite the first three Rules of Normalization without peeking, you don't need this chapter anyway. On the other hand, you don't really have anything to lose by learning this stuff now, before you get everything all muddled up, do you?

I say that it holds *information* rather than *data*, because the documents in it are organized, sorted, and stored according to a model I came up with before I started putting documents in its drawers.

It's All About the Data Model

Data becomes information when it is organized. Data by itself, like the hodge-podge of papers in my cardboard box, is not terribly useful. Moreover, it's not terribly important what type of container you keep the data in.

If I simply open a drawer of the cabinet and empty the box into it once or twice a month, I am no further ahead than if I just keep bringing in more boxes to shove under my desk. In fact, I don't really need the filing cabinet at all. I could organize a set of cardboard boxes the same way the drawers in the cabinet are organized because I have a method of organization, or a *Data Model,* independent of the container in which the information is stored.

> A data model is a logical description of things we are interested in within a specific situation. It consists of statements defining objects and events, identifying pertinent facts about those objects, and explaining how they relate to one another. Unlike a plastic model of an airplane, a data model is not a physical representation of those things. For that reason we will avoid talking about tables in an Access database, which are the "physical" objects with which a database works, until we have a more-or-less complete data model in place.

That's an important point to keep in mind while building your database. Your data model, *or the way you organize the information in which you are interested*, is independent of the container in which you will store your information. If it's properly designed, the same data model can be incorporated into an Access database, a SQL Server database, or any other relational database application.

Okay, Cardboard Box Guy, Model This

We're ready to look at the process by which you create a Data Model and I think my filing cabinet can work as a very simple example to get us started.

What Do I Want to Keep Track of—the Entities

My filing cabinet contains all kinds of documents: canceled checks, credit card statements, bank statements, bills from our doctor and dentist, renewal notices from our insurance company along with Declarations Pages for our policies, and mortgage papers from when we bought the house. We keep some of them for legal reasons (tax returns, for example) and others for historical reasons (insurance documents, for example). All of these documents are important enough for our family to want to keep them in a safe place where we can refer to them when necessary. These documents are the *entities* around which our filing system is built.

In database terms, an entity is an object in which an organization is interested and about which the organization wants to collect and maintain information. To be an entity, a thing must exist and be distinguishable from other objects. It can be *concrete*—like the pay stubs in my filing. You can feel and see them. It can be *abstract*—a fact or concept like a "forty hour work week". You can't see a "forty hour work week", but we all know what it is, at least in theory.

We define entities by stating what kind thing they are and how they are different from similar things of that kind.

Example: A DOCUMENT is a PAPER or GROUP OF PAPERS that contains information retained for legal or historical reference.

In this case, it was fairly easy to figure out what kind of thing a DOCUMENT is; it's a piece of paper or a group of papers that contains information. It was only a bit harder to describe how a DOCUMENT is different from other pieces of paper that contain information—a newspaper, for example. For this data model, DOCUMENTS are retained for legal or historical reference; other pieces of paper with information on them are not. In another context, my data model might need to define DOCUMENT differently.

Tech Talk—Formal Entity Definition

A formal Entity Definition consists of three parts:

- *Term* to be defined
- *Class* of things in which it belongs
- *Differences* it has with other members of the class.

We can recursively define Entities and Classes to the level of generality or specificity required by the data model we're creating. In other words, a Class is defined using the same formal definition; an Entity defined in one definition can be the Class by which another Entity is defined in a more specific definition. See Table 3-1.

Formal Entity Definitions		
Entity	**Class**	**Difference(s)**
PAPER	MATERIAL	Made of cellulose pulp
PRINTED PAPER	PAPER	On which information is printed using mechanical or manual means
DOCUMENT	PRINTED PAPER or GROUP OF PAPERS	Retained for legal or historical reference

Table 3-1 Formal Entity Definitions for PAPER, PRINTED PAPER and DOCUMENT

Enough Entities Already

Obviously, you don't need to go on defining entities indefinitely. It is only necessary to create enough definitions to fully account for all of the entities of importance in your data model. And, of course, at some point you will reach a level of sufficient generality that it makes no sense to include the definition in your data model.

For example, while DOCUMENT is an entity of interest in my cardboard box database, PAPER and PRINTED PAPER are not. MATERIAL is obviously too general to be of any interest in this data model. However, we will definitely need to continue defining additional types of documents (i.e., members of the Class, DOCUMENT), such as BANK DOCUMENT, INSURANCE DOCUMENT, TAX DOCUMENT, etc.

Hierarchies and Entity Groups

When I start pulling papers out of my cardboard box in preparation for filing them in my filing cabinet, I don't immediately transfer them. First, I sort them by groups: utility bills in one pile, bank records in another, and insurance papers in still another pile. These piles represent an intermediate classification or category of interest to me. Later, when I need to find a DOCUMENT in my Filing Cabinet, those DOCUMENT GROUPS help me focus my search.

For example, if I'm looking for cancelled checks, I start in the section of the drawer that holds BANK DOCUMENTS.

It really isn't necessary to group DOCUMENTS this way to file them in the filing cabinet, but I do it because *it helps me find them a little quicker later on*. That's an important point and we'll come back to it later.

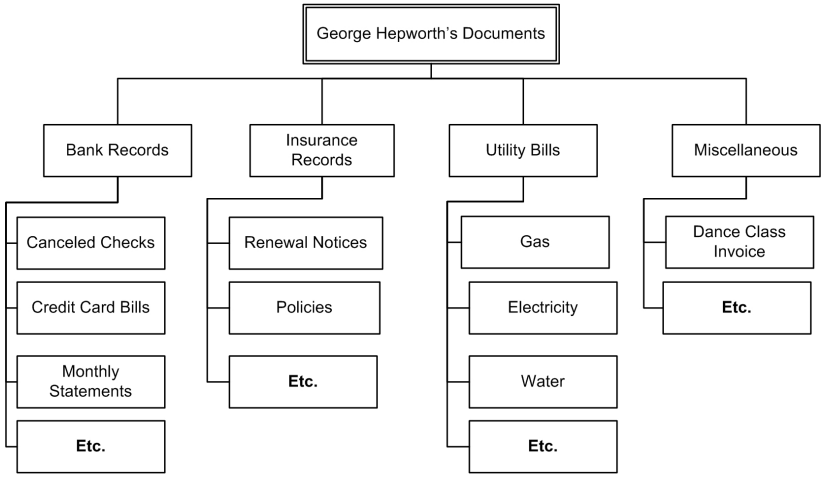


Figure 3-1 Three-level Hierarchy of Document

Figure 3-1 illustrates the three-level hierarchy used in my filing cabinet database (with a lot of detail left out to save space).

As this model of the data in my Filing Cabinet Database shows, George Hepworth's Documents includes DOCUMENT GROUPS—Bank Records, Insurance Records, Utility Bills, and Miscellaneous. Within each DOCUMENT GROUP, there are one or more DOCUMENT TYPES.

For example, the DOCUMENT GROUP called BANK RECORDS includes DOCUMENT TYPES called CANCELED CHECKS, CREDIT CARD BILLS, MONTHLY STATEMENTS, and so forth. The DOCUMENT GROUP called UTILITY BILLS includes GAS, ELECTRICAL, WATER, and others.

In database terms, a collection of similar entities, such as UTILITY BILLS, is an entity GROUP—a group of entities of the same type. The Dance Class Invoices in my filing cabinet comprise another entity GROUP called DANCE CLASS INVOICES; INSURANCE RENEWAL NOTICES are another.

Not All Relationships are Hierarchical

So far, the relationships we've seen between DOCUMENTS in my Filing Cabinet Database are the hierarchical ones in Figure 3-1. As you can see, these relationships group related entities into more specific sub-groups, branching out like the roots of a tree. But there are other kinds of relationships between entities and that is where a Relational Database Management System like Access comes into play. You'll learn more about relationships later in this chapter and in the next. For now, the important thing to remember is Access is a relational database, rather than a hierarchical one, and this gives us some important benefits when we begin converting our data model into tables in the database.

For this particular data model, another fact in which I'm interested is the TIME PERIOD to which a particular DOCUMENT applies. There are actually two ways to incorporate TIME PERIOD into the data model we've been developing for my Filing Cabinet Database.

Group by DOCUMENT GROUP, then by TIME PERIOD

In this data model, DOCUMENT GROUPS would take precedence over the TIME PERIOD to which they apply.

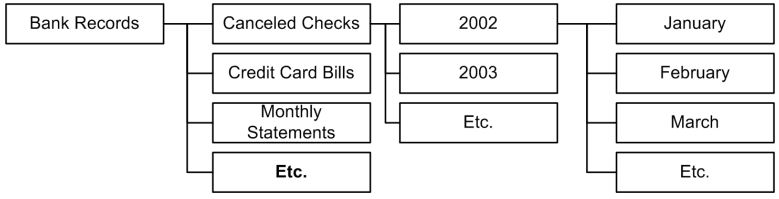


Figure 3-2 Sort by DOCUMENT GROUP By TIME PERIOD

To find a canceled check for March 2002, I would first go to the section of the filing cabinet where I keep BANK RECORDS, then find CANCELED CHECKS, then CANCELED CHECKS for 2002, and then CANCELED CHECKS FOR MARCH.

Sort by TIME PERIOD, then by DOCUMENT GROUP

In this data model, TIME PERIOD would take precedence over DOCUMENT GROUPS.

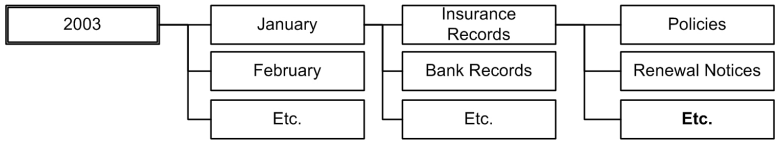


Figure 3-3 Sort by TIME PERIOD by DOCUMENT GROUP

To find a RENEWAL NOTICE for my auto insurance, I would start by looking in the drawer with records for the year, 2003, then the month, JANUARY. In the JANUARY section I'd find INSURANCE RECORDS for JANUARY and, finally, RENEWAL NOTICES.

Choosing a Model—Workflow and Business Rules

Even though this is a very simple database, there are two possible ways to define the relationships between DOCUMENTS, DOCUMENT GROUPS, and TIME PERIODS. Therefore, we have to choose between two data models. Which is the right one?

I've had to address this question in the process of designing every database I've ever built. It is almost always possible to express relationships between certain entities in more than one way. Often, different approaches can be equally valid on the surface.

Therefore, I believe the answer to that question is, "Use the model that best serves the purpose of the database." This choice is only partly dependent on the data itself. Selecting the right data model means carefully evaluating the purpose for which the database will be used and the workflow it will support. Who will use it and how will they use it? What rules apply to using it? And that, in turn, requires you to spend enough time interviewing your customers to fully understand the purpose of the database and the workflow it will support.

The Customer is Always Right

Your customers include the managers or executives who requested the project, the subject matter experts who understand the business processes and workflow which it will support, the end users who will use it as part of their regular workload, and the external customers for whom your organization provides goods or services. Each of these customers is important to the success of the project and each has something valuable to contribute to that success.

Your organization may use the term *Stakeholders* instead of *Customers*. I prefer *Customer* because it helps me remember that I am providing a product for which someone is paying me.

- Managers, of course, hold the decision-making power over your project, so their support is essential. However, that's not the only thing they bring to the table. Their position usually provides them with a broad view of your organization's long-term goals and plans. Their insight can help you make good decisions about the role your database will play in those over all plans.
- Subject Matter Experts (SMEs) are the people with technical expertise and knowledge about the business itself and the workflow your database will support. Working closely with them to share their knowledge is another crucial element in your success.
- End Users are the folks who have to use your database. If they aren't happy with it, they will do their very best to avoid using it, up to and including active sabotage. Make sure they are happy. Make every effort to find out how they do their jobs, what words they use to describe what they do, who provides them with input, and who gets the output they produce. The more you know, the more likely you are to produce a database they will actually use.
- External customers may be directly or indirectly affected by the database you produce. Therefore, the degree to which you account for their needs will vary. When there is some direct interaction, you need to do your best to make it as smooth and flexible as you can. For example, if your database will be used by a Customer Service Rep to look up external customer records, a key factor in providing good service to customers will be making it as easy as possible for your CSRs to find their records.

Choosing My Filing Cabinet Data Model

I don't have to spend a lot of time interviewing myself to find out how I want to use my Filing Cabinet. It was fairly easy for me to pick the data model I want to use, DOCUMENT GROUP-DOCUMENT-TIME PERIOD, the one in Figure 3-2. Here's how I decided.

The way I *retrieve* information is more important, in this case, than the way I *enter* it.

Most of the time, I need to look for a particular DOCUMENT in a particular DOCUMENT GROUP: I want to compare my current electric bill to last year's bill for the current month, or I have a question about the current coverage limit on my homeowners insurance. In the case of bills I receive on a regular cycle, like the electric bill, I could easily get to the right information under either data model because I should find the appropriate DOCUMENT in any TIME PERIOD in which I choose to look— July, 2002 and July, 2003, for example.

However, in the case of my insurance coverage, the DOCUMENT GROUP-DOCUMENT-TIME PERIOD approach is clearly more efficient. For one thing, the policy renews once each year, but I don't remember the renewal date off the top of my head. If I used the TIME PERIOD-DOCUMENT GROUP-DOCUMENT data model, I would have to search through each TIME PERIOD until I found the one with that DOCUMENT in it. Even with a good starting guess, the chances are that I'll have to look in more than one month to find it.

For another, there is only one month associated with each insurance renewal. Once I get to the section of the Filing cabinet where my insurance records are, I'll probably find only one TIME PERIOD there, the one in which the renewal resides. I only have to store a TIME PERIOD if there is something to put in it.

Granted, my little Filing Cabinet is nowhere near as complicated as the Access databases you'll be building in the days and weeks ahead. You will, however, be making decisions about your own data models using processes and principles like these.

Now, let's learn a little bit more about the entities we'll be tracking.

Entities Have Attributes

When we talk about an entity, we are referring to the thing itself. The tables in your Access database will be based on entities. We are also interested in facts about entities. For example, one thing we want to know about a DOCUMENT is the kind of information it contains. We called these DOCUMENT GROUPS. Another thing we need to know is the TIME PERIOD to which it applies.

In database terms, attributes are facts about an entity in which we are interested. To put it another way, one purpose of the database is to capture attributes about the entities in the database.

For the entity DOCUMENT, for example, we have identified two attributes: what *kind of information* it contains and the *time period* to which it applies. In the data model, we called these the DOCUMENT GROUP and TIME PERIOD.

For any given attribute, an entity has only one value. If I have a bill from the gas company, for example, it can only be for gas I consumed during one specific period, such as May 15 to June 15, 2003.The gas company is not free to send me bills for arbitrary, overlapping, or duplicate time periods, or for completely unrelated services such as phone lines to my house. It may seem like I'm only stating the obvious, but when it comes time to create your Access tables, you'll need to be able to count on this one-to-one relationship between attributes and the values they can have.

When we create physical tables in our database, the attributes will become fields in the tables representing the entities.

Enough Attributes

Obviously, we are not interested in capturing every possible fact about an entity, just as we weren't interested in including every possible entity. I don't care, for example, if my bank statements come on blue paper or pink paper, or that my bills from the phone company list international calls separately from other long distance calls. Those are attributes that I don't need or want to keep track of for the purposes of this particular database. However, it is important to the success of your database that you do identify all of the attributes that will be of interest to you before you start creating tables, or as many as possible. I've seldom had the intelligence (or good luck) to identify all of the relevant entities and attributes on the first try. However, the point is always to be as thorough as possible as early as possible in the development process.

Tech Talk—Entities, Attributes and Relationships

So far, I've kept my discussion of entities, attributes, and data models informal because I wanted to help any novices among us grasp the basic concepts before tackling the jargon that goes with them. In the process, I have over-simplified some concepts and omitted others.

So, here's a more formal discussion of what we've covered so far. To do so, though, we'll need to use examples that are somewhat more complex than my cardboard box filing system. It's served its purpose in getting us started, but it will now be left behind.

Entity

An entity is an object or event that exists. It can be differentiated from other objects. For instance, the PC sitting on the floor under my desk, Serial Number 07130-2M2-0120, is an entity. It can be uniquely identified as one particular PC in the universe, although it was part of a production run of hundreds of identical PC's.

An entity can be concrete—a canceled check, for example—or abstract; a concept such as contract, or an event such as transaction. For example, the transaction in which I purchased the PC was an event that occurred on October 21, 2001. The transaction included a salesperson at a local computer store, my credit card, and me. The purchase event itself is an entity. The date, the PC, the salesperson and myself are part of the purchase event.

Entity Sets

An entity set is a group, or set, of entities of the same type, such as all of the canceled checks from my personal checking account. They are all one kind of thing, but are unique and distinguishable from each other in several ways.

Common Members

Two or more entity sets can have members in common. For example, the entity set *school employee*, referring to everyone who works at my daughter's school, and the entity set *school parent*, referring to parents of students in the school, can and do have members in common. Some school employees have children who attend their school. The database term for this principle is that entity sets *need not be disjoint*.

When you are deciding which Access tables you'll need to represent entities, you'll learn more about the importance of this principle.

Attributes

Each entity has a set of one or more attributes. For most entities, such as school parent, there is a fairly well-defined set of attributes: for example, name, street, city, state and zip, home phone, and work phone.

As I noted in the informal discussion earlier, it is usually possible to identify more attributes than are of interest within a data model. For example, while school parents are either *male* or *female*, that attribute may or may not be of interest in a database of school parents.

You don't know that, of course, until you've interviewed the customers who will use it. If there is a business reason for keeping that information, you'll need to include it.

However, in my experience, the larger problem in most development projects is failing to include all of the relevant attributes in the initial data model. In other words, it's better to start with a list of all possible attributes and whittle it down, rather than to have to incorporate an overlooked attribute at a point where doing so causes major and, therefore, expensive changes in an existing database.

Domain

Domain refers to the set of permitted values for an attribute. For example, there are only two permitted values in the Domain of Gender, male and female. For ZIP CODE, the Domain is either five or nine positive integers.

Attribute/Data Value Pairs

For each entity in an entity set, there is a set of one or more pairs of attributes and data values; for each attribute, there is one attribute/data value pair. (Nope, that isn't a foreign language, although it might seem like it at first.)

This is a formal way of saying what we've already talked about: an entity has one or more attributes—facts about the entity of interest to our organization. We're simply expanding on this by adding the requirement that, to fully and accurately describe an entity, the attribute must be with an appropriate data value.

For example, consider a shirt. It has several attributes, including size, whether it has buttons, whether it has long or short sleeves, what material it is made of, and what color it is. COLOR, therefore, is an attribute of shirt. However, it would make no sense to refer to the shirt's color without specifying what that color is: red, blue, pink, etc. Red, blue, and pink are the data values for COLOR for this particular entity, SHIRT.

Later, when you learn about the rules of normalization, we'll reinforce the importance of this point. For now, just remember each attribute consists of an attribute and its corresponding data value.

Here's another example. A particular individual in a mailing list database is described by the set of attribute/data value pairs illustrated in Table 3-2:

Attribute	Data Value
FirstName	William
MiddleName	Henry
LastName	Carterfield
NickName	Bill
Birthdate	08/10/1980
Anniversary	6/30/2001
RelationshipType	Friend

Table 3-2 Attribute-Data Value Pairs

As Table 3-2 shows, each attribute specified for one *individual* has one, and only one, corresponding data value.

Relationships

Up to now, I've used the terms relationship and relational in an informal way. I briefly talked about the hierarchy of DOCUMENTS in the course of discussing how we came to group documents for filing. That is one type of relationship, but not the most important type of relationship in a Relational Database such as the ones you will build with Access.

Relationship

A relationship is an association between entities in different entity sets. For example, consider HOUSEHOLD and INDIVIDUAL, which are entity sets in a database of school families. In other words, this database keeps track of households associated with the school and individuals associated with those households.

In that database, an INDIVIDUAL is any person in whom the school has an interest: a school employee, a parent or another relative of a student in the school, or a student in the school. A HOUSEHOLD is defined as one or more persons who share a single address.[5]

[5]Although it's not critical to this discussion, it is interesting to recognize that this definition is broad enough to include households consisting of a married couple, a single person, two (cont.)

The relationship between HOUSEHOLDS and INDIVIDUALS is that each HOUSEHOLD consists of, or is made up of, one or more INDIVIDUALS who live at the same location.

For the moment, let's assume that, in this database, an INDIVIDUAL can only reside in a single HOUSEHOLD. We'll consider the opposite situation later in this section.)

Role-based Relationships

We can define the *relationship* between HOUSEHOLD and INDIVIDUAL by specifying the *role* each plays in the relationship between them. Again, for the purpose of this relationship, the role of the INDIVIDUAL is *resides in*. The role of HOUSEHOLD is *residence for*.

An INDIVIDUAL is a person who *resides in* a Household.

A HOUSEHOLD is a *residence* for one or more INDIVIDUALS.

In a binary relationship, like the one between HOUSEHOLD and INDIVIDUAL, it is possible to state the relationship in terms of either member.

Here are a few other examples of entities involved in this kind of role-based binary relationship.

A CUSTOMER purchases a PRODUCT.

A PRODUCT *is purchased by* a CUSTOMER.

A STUDENT *is enrolled in* a CLASS.

A CLASS *consists of* one or more STUDENTS.

Descriptive Relationships

Some relationships are descriptive, rather than role-based. For example, in the attendance portion of our school database, a STUDENT can be absent or tardy on one or more occasions. The two entities are STUDENT and ATTENDANCE RECORD. The relationship between them is that a STUDENT *is absent or tardy* on a particular date. ATTENDANCE RECORD captures that information by describing or specifying the date(s) and attendance status of each STUDENT.

unmarried partners, a parent and grandparent, and other permutations involving at least one person.

Relationship Set

A relationship set is a set of relationships of the same type that connects entities in one entity set with entities in another. Perhaps the best way to explain this is with an example.

The RELATIONSHIP between CUSTOMER and PRODUCT, as stated above, is:

A CUSTOMER *purchases* a PRODUCT.

Therefore, if we have a group of CUSTOMERS—John, June, and Jerry—and a group of PRODUCTS—a red bicycle, a blue bicycle, and 2 green bicycles—the RELATIONSHIP set between them could be:

John purchases the red bicycle.

June purchases the blue bicycle.

Jerry purchases a green bicycle.

Although the other green bicycle is in the PRODUCT group, it won't participate in a relationship with a CUSTOMER until the customer purchases it. It is not a member of that relationship set until that happens.

Relationship Types

We are primarily interested in three types of relationships. These are:

- One-to-One
- One-to-Many
- Many-to-Many

One-to-One Relationships

In a one-to-one relationship, each entity in one entity set is related to one, and only one, entity in a second entity set. For example, many corporations provide company cars for their employees. There are two ways for the company to do this, depending on which business rule they've adopted.

Business Rules

The business rules of the company determine how cars are allocated and the appropriate relationship is determined by those business rules. First, company XYZ maintains a fleet of cars that employees check out as

needed. (This leads to a many-to-many relationship, which we'll come back to a bit later.)

Second, a company can assign one car to one employee for his exclusive use. For example, the ABC Company provides each of its sales managers with a car. This is a one-to-one relationship, assuming, of course, no sales manager is allowed to have two company cars at the same time (another business rule).

A SALES MANAGER *is assigned* a COMPANY CAR.

A COMPANY CAR is assigned to a SALES MANAGER.

If you were creating the data model to manage this company's company car fleet, you would need to create a one-to-one relationship between SALES MANAGERS and COMPANY CARS.

One-to-one relationships are the least common. Moreover, when it is time to create Access tables from your data model, most one-to-one relationships can be put into one table. So, pay close to these relationships in your data model to be sure you set up your tables correctly.

One-to-Many Relationships

In a one-to-many relationship, each entity in one entity set is related to one or more entities in a second entity set. For example, a potato grower can cultivate one or more fields of potatoes. This is a one-to-many relationship. If you were building the data model for a government agency tracking pest control for local potato farmers, your data model would need to accommodate this relationship.

A POTATO GROWER *cultivates* one or more POTATO FIELDS.

A POTATO FIELD is cultivated by one POTATO GROWER.

One-to-many relationships are the most common and usually the easiest to define and set up in a database. Here are some other examples of one-to-many relationships in some of the databases I've built for customers over the years.

An INSURANCE AGENCY *employs* one or more INSURANCE SALESPERSONS.

An INSURANCE SALESPERSON *is employed by* one INSURANCE AGENCY.

A CLASS *enrolls* one or more STUDENTS.

A STUDENT *is enrolled* in one CLASS.

In the last one-to-many example above—a STUDENT in a CLASS— the business rule for this elementary school requires a STUDENT to be enrolled in one and only one CLASS during each enrollment period, which is the SCHOOL YEAR. For example, Josefina Carillo is enrolled in the seventh grade at this school. The seventh grade includes Josefina and 22 other students: one class, many students.

A community college, with different enrollment and curriculum requirements, has a different rule about enrolling in classes. The community college allows students to enroll in several classes during the enrollment period, which is the SEMESTER or QUARTER. Each class is independent of the other classes the college offers, so each student has his own unique schedule of classes.

For example, John O'Hara has classes in Math, English, and Biology, while Keiko Yamasaki has classes in Math, Botany, and Electrical Engineering. This is an example of the last type of relationship we'll discuss, Many-to-Many.

Many-to-Many Relationships

In a many-to-many relationship, each of the entities in one entity set is related to one or more entities in a second entity set. Each of the entities in the second entity set is related to one or more entities in the first entity set.

For example, as we saw earlier, Company XYZ's business rule about company cars says any car in the car pool can be checked out by any employee, which creates a many-to-many relationship.

> An EMPLOYEE *checks out* one or more COMPANY CARS.
>
> A COMPANY CAR *is checked out by* one or more EMPLOYEES.

Of course, there is another business rule that says an employee can only check out one car at a time, but on any given day, any car in the company car pool can be checked out by any employee: Many cars, many employees.

In the case of the community college classes, any student can enroll in one or more classes during each semester. A class can enroll one or more students.

> A CLASS *enrolls* one or more STUDENTS.
>
> A STUDENT *enrolls in* one or more CLASSES.

Here is another many-to-many relationship I encountered while building a database for an elementary school database.

A STUDENT *belongs to* one or more HOUSEHOLDS.

A HOUSEHOLD *includes* one or more STUDENTS.

This one may surprise you at first, but if you think about it for a moment, you'll see how this can, and frequently does, happen. Not that many years ago, the traditional HOUSEHOLD was presumed to be two parents and one or more children. Today, a HOUSEHOLD can be a single parent and one or more children, an unmarried couple and one or more children, or a parent, one or more children, and a grandparent.

Many students don't live exclusively in one HOUSEHOLD. They spend some time in Mom's house and some time in Dad's house. The attendance database I built for this school, therefore, needed to accommodate a many-to-many relationship between STUDENTS and HOUSEHOLDS, based on the real world circumstances our data model revealed.

Business Rules and Data Models

Each organization has its own business rules about how it conducts its operations. With company cars and sales managers, schools and classes, potato growers and potato fields, and each of the other relationships you'll encounter, you'll need to know not just what entities are involved, but how your organization applies its rules to those entities.

In almost every situation where you are defining relationships between the entities in your model of data that your database will track, you'll have to consult your subject matter experts, end users, and managers to learn the business rules that apply. This is the only way you'll be able to create an accurate, usable database.

Business-Specific Entity Definitions

In the example of the community college, it is obvious that it needs to accommodate multiple students in multiple classes, but it can't, or shouldn't, allow students to sign up for more than one class during the same time on the same day. To achieve this result, the college must have a different definition of CLASS than the elementary school, one that includes week days and times as well as enrollment period.

Key Attributes

So far, you've learned that the business of creating a relational database depends on three things.

- Identifying the entities of concern to us
- Identifying and listing their relevant attributes
- Identifying the relationships that link our entities with one another.

To complete the process, we need one additional piece of information, which is the *key attribute* that uniquely identifies each entity in each entity set.

Normally, a key attribute is one of many attributes of an entity. For example, the serial number on the back of my PC, 07130-2M2-0120, is unique; no other PC has that identical serial number. The PC has other attributes of potential interest, such as the CPU speed, amount of RAM it holds and the storage capacity of its hard drive. However, it is that serial number that uniquely differentiates it from all other PCs with similar attributes.

As an individual, I can be uniquely identified only by referring to a combination of attributes. My names, George, Russell, and Hepworth are not individually unique, and probably not even unique in that combination, although I've not met anyone with the same name yet. My social security number is very close to being useful as a unique identifier, especially when combined with my name. In a logical data model, then, we can consider the *combination* of first name, middle name, last name and SSN enough to uniquely define a person.

With these two key attributes, we can uniquely identify the relationship between any two entities, PERSON and COMPUTER EQUIPMENT.

PERSON ID	owns	PC Equipment ID
George Russell Hepworth SSN 555-55-5555	owns	07130-2M2-0120

Table 3-3 Relationship Example

As you might expect, when you convert your logical data model into physical tables in an Access database, one of the most important steps will be setting up key attributes for entities so you can uniquely identify the entities in each table. In a logical model, key attributes are usually

fairly easy to identify, but that task can be a bit more challenging when it comes time to create the physical tables.

What is not obvious at this point is you will seldom, if ever, use one of these real world or natural key attributes as the primary key in a physical table. Consider, for example, the problem we've identified with names and social security numbers. In our logical data model, it is the combination of these four attributes that uniquely identifies a person. Access would allow you to set them up as a compound key in a physical table, but that would quickly lead to a lot of problems, problems you'll want to avoid as much as possible.

SSN by itself is more attractive as a key attribute and, in some contexts, it is adequate as a key attribute in a *logical* model. Still, for reasons you'll learn in Chapter Four, it doesn't quite work as the primary key in a physical table.

Further Study

The foregoing discussion of entities' attributes and relationships is really no more than an introduction to a fascinating area of study. My intent is not to prepare you to take on a serious data analysis project. Rather, I want you to be comfortable with the concepts of entity, attribute, and relationship, so you'll be ready to start building well-designed Access tables. If, like me, you're fascinated by the topic, you'll find a list of references for further study in Appendix A.

Creating an Informal Data Model

By now, you should be ready to create your first, informal, data model. Let's start with one that is useful, but fairly easy to set up: a phone number, email, and address database for family and friends. To make it more interesting, I also want it to keep track of birthdays and anniversaries.

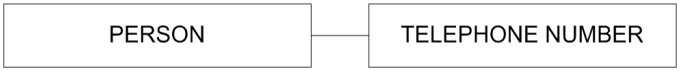
Your Data Model

Start by listing all of the things you think you'll need to know to set up the database, things like names, email addresses, mailing addresses, and phone numbers. Don't look at my list on page 3-26 until you've written as many as you can think of yourself. On the following page, list your entities and attributes, and key attributes (the ones that uniquely identify each entity.

Now, on page 3-23 (or on a notepad if your prefer), sketch out the relationships between your entities. Here are some simple conventions to get you started.

Enclose your entities in rectangles. 

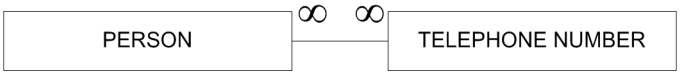
Draw lines between related entities. There is a relationship between persons and telephone numbers, for example.



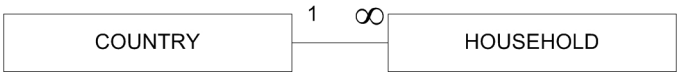
Relationships can be one-to-one, one-to-many, or many-to-many. To indicate these relationships between your entities, use a "1" for the one side and the symbol "∞" for the many side.

Because each person can have one or more phone numbers, we place the ∞ next to phone number. Because one or more persons can use each phone number, we place the ∞ next to person as well.

This, as you know, is a many-to-many relationship.



Here is an example of a one-to-many relationship.



Some of my family lives in the United States and some of them live in another country, Venezuela to be precise. That may or may not be an important feature of *your* database. For *my* purposes, it is important, because there are different formatting requirements for telephone numbers and postal codes—in fact there are no postal codes in Venezuela. By definition, a household consists of a group of people living at a single location, so a household can only be in one country. Many households, on the other hand, exist in both countries.

If there isn't enough room on the following pages for your sketch, or if someone else has already used then, take out a yellow pad and start drawing boxes and lines.

Sketch out your Entities and Relationships here.

My Data Model

Let's look at my model for a mailing address, email and phone number database. While I'm explaining what I did and why, you can compare it to your model and the choices you made in creating it. The following discussion will be fairly detailed at some points, probably more detailed than would otherwise be justified for a database as simple as a mailing, phone and email list database. At the same time, I'm going to skip over some of the more technical aspects of data modeling.

To repeat a point I made earlier, this book *is not* intended to make you a fully qualified data modeler or database developer. It *is* intended to illustrate principles I feel are important to grasp before you begin creating the physical tables in the database. One simple example alone won't achieve that goal, although I have tried to pack as much into it as it will bear.

Entities, Definitions, and Attributes

You may have included more or fewer entities than I did and more or fewer attributes for each. Based on the purposes for which this database is intended, I think you should include at least the things I've identified.

As you recall, I defined the purpose of the database to be mailing, email, and phone numbers *along with birthdays and anniversaries*. If I had defined the purpose of the database to be just the mailing, email and phone numbers, I would have included less information, although most of the data model would have been the same.

Since this is quite likely your first attempt at creating a data model, don't worry if you didn't get it quite right on the first try. You'll get better with practice.

In my mailing and phone list data model for family and friends, I am interested in the following entities and attributes. The table on the following page also includes definitions for the entities listed along with an example of each.

Entity	Definition	Example	Key Attribute(s)
Household	Group of people who reside at a single location	Tom & Cindy Hepworth household	Household Name, Head of Household Location, Country
Person	Individual for whom I record a phone number, email or mailing address	Tom Hepworth	First Name, Last Name
Address	Location at which a particular organization or person may be found or reached	2121 E. Haverford St. San Francisco, CA 94107 USA	Street, city, state, postal code
Phone Number	Number assigned to a telephone	415 111-1010	Country code, area code, prefix, extension
Email Address	Electronic address to which electronic messages are delivered	tomh@myco.com	email address

Table 3-4 Entities and Key Attributes

In addition to the key attributes listed above, I am interested in nicknames of friends and family members, any suffixes (e.g., jr.) birth dates, anniversaries and deceased dates of people, what the relationship between each person and myself is (friend or family member), what type of address I have for each person (work or home, PO Box or street address), what type of phone number I have for each person (home, work, cell phone, fax, pager) and what type of email address I have for each person (work, personal, etc.).

Here's my sketch of the entities and relationships.

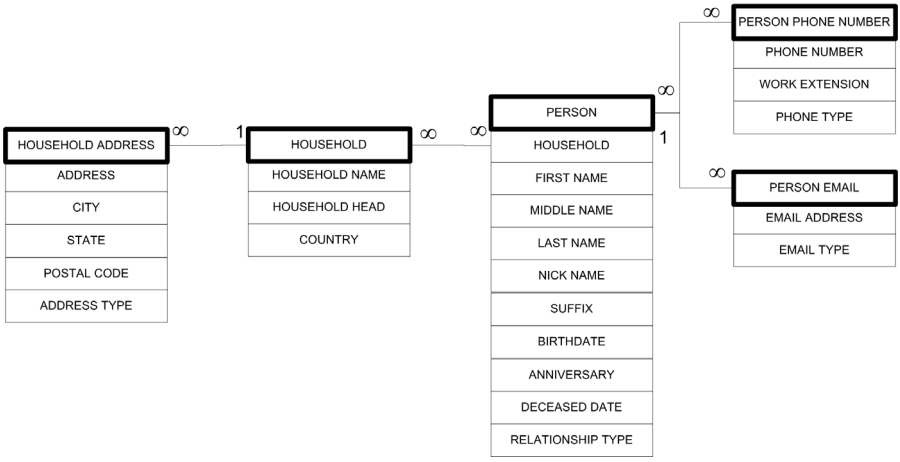


Figure 3-4 Logical Data Model—Phone, Email and Mailing List

I decided that, in addition to the country where a Household is located, I want to have a household name and head of household for each household. The household name is useful when I want to send a Christmas card to all members of a household, for example.

Hedging my Bets

I've included a couple of things in my data model, even though I'm not yet sure I'll later use them in the database. I often do that during the data-modeling phase. In my experience, it is usually a lot easier to go back and delete something if I decide I don't need it, than it is to try to insert an overlooked entity or attribute into a half-finished database.

For example, being able to identify a head of household strikes me as something I want to be able to do, even though I can't think of a way to use it at the moment. I'll keep it in until I make up my mind about its importance.

Managing Data—Delete vs. Inactive

I also decided to include the date when a family member is deceased, partly because it is somewhat relevant to the purpose of this database and partly to make a larger point about managing data and table design. On one level, this may be somewhat trivial in a phone number, email, and mailing list. However, it is worth discussing as it helps illuminate an important principle in database design.

I decided this database would include birthdays and anniversaries. I figured that would allow me to display a list of August birthdays, for example, and quickly look up the appropriate address or addresses at the same time[6]. While I was writing down all of the things about my family and friends it occurred to me that life events, such as births, marriages, divorces, and deaths occur in every family. We celebrate the positive events, like birthdays and anniversaries, and I want them in the list so I can acknowledge them at the proper time. This is, in fact, one of the reasons for making this a database.

On the other hand, I'm not at all sure I'm interested in divorce dates, so I didn't include that information in my model.

Death, however, is a bit more complicated. If someone in my list were to die, I could just remove that person from the list. I don't want to do that, and not just for sentimental reasons. The principles of good database design tell me that I don't want to get into the habit of removing records from a database.

Remember back to Chapter two, when I was discussing the reason to choose Access over Excel for a particular purpose? One of the key factors I mentioned was that a database is good at keeping historical, as well as current, information. On principle, therefore, I don't want to remove data that has some sort of historical significance. I figure information about family members who have passed on falls very definitely in that category. Instead of deleting them, then, I want to include a piece of information that will identify their status. In the database, deceased date will prevent me from accidentally including them on active lists for birthday and Christmas cards.

Status

Deceased date is a specific example of a generally useful piece of information found in nearly all databases: *active and inactive status*. Anytime you have an entity whose status can change over time, you can best handle it by recording the date(s) that status changes rather than deleting or adding new records.

[6] I know, Outlook does the same things, maybe even better than my database can do. You probably don't need this database any more than I do. That said, it's still worthwhile designing it as an exercise in learning how to design and built a database.

Who's Related to Whom

Although this is not the only way to do it—and you may have decided otherwise—I decided *addresses* are related to households while *phone numbers* and *email addresses* are related to individuals. Here's how I arrived at that conclusion.

If I ask one of my friends or family members for their address so I can send someone in their household a birthday card, they will all give me the same address, the street address where they live or the P.O. Box where they get their mail. My brother, his wife, and each of their kids will all give me the same address in California, for example.

However, that isn't true if I ask for their phone number. They'll usually respond with their own question, "Do you want my work phone or home phone?" And increasingly, they'll mention a cell phone as well.

The same is even truer for email addresses. Nearly everyone I know has their own email address; most of them have more than one. It is rare, however, to find two or more people sharing an email address.

So, when I set up this database, I decided I would relate mailing addresses to households and email addresses and phone numbers to people.

Gray Areas, Trade-Offs, and Lessons Learned

I could also make a case for treating addresses the same way as I did phone numbers; that is, they are related to individuals rather than households.

Some people in my database are actually business associates rather than friends. The only address I have for them is a work location; I don't know their home address. Moreover, I don't know any of the other household members at that address. In that sense, their addresses can be seen as being related to the person rather than the household.

The opposite is true for most of my family members; I have their home address but not a work address, and I am interested in all of the household members at that address.

In both cases, the applicable business rule is, "When adding new persons to the database, record the address to which you are most likely to send mail: home addresses for family and friends, work addresses for business associates." That's because my mailing list is set up to help me send birthday cards, Christmas cards, or other, similar letters and cards to family and friends. I only need one address per household to do that.

The opposite is true of phone and email lists. When I want to get in touch with someone by phone or email, the choice of numbers or email addresses does matter. During the workday, I want to call the person's work number or use their work email, but during the evening or on the weekend, I want to use their home number, cell number or personal email address.

Relationship Types

The relationship between persons and phone numbers is many-to-many and, for some phone numbers, that's true most of the time. For example, most home phones are shared by all members of the family (although those of you who have teen-agers may feel otherwise) and each of those family members uses more than one phone.

Other phones are used exclusively, or almost exclusively by one person. For example, most cell phones are personal phones. There is clearly only one person in that relationship. Phones, like addresses, are a bit more of a gray area than might first appear.

Lessons Learned

I am spending a good amount of time on this subject, not because I'm particularly obsessed with phones, email and mailing addresses, but because deciding how to handle them in your data model is a special case of a more general issue that frequently comes up when it is time to convert your logical data model into physical tables in Access. Sometimes, you just have to make a choice between two valid approaches.

The decision is based, as much as anything, on the way the data will be used and the business rules that apply to that use. And, as I've stated before, this requires you spend considerable time with your customers, gaining an understanding of their workflow and requirements.

The important thing is that by creating, evaluating, and revising your data model in light of the definitions and business rules you discover, you will better understand how all of the relationships work, or should work. You will be able to make informed decisions about the potential advantages and disadvantages of each approach. Doing so will help you head off problems down the road.

Trade-Offs

The trade-off in my decision to create the relationship between persons and phone numbers instead of between households and people is that I have a many-to-many relationship between them. In a database, many-to-many relationships are almost always more complicated to handle than one-to-many relationships. However, by going that route, I gained something else: an easier way to relate people with the phone numbers I am likely to use to contact them.

Summary

In this chapter, I've given you an overview of the initial phase of creating a new Access database, starting with the cardboard box-to-filing cabinet metaphor. You learned that data becomes information when it is organized according a logical (as opposed to physical) model.

You saw that before you can transfer your raw data from a cardboard box into a filing cabinet, you need to spend time analyzing the bits and pieces of data that make up the information in which you are interested. You need to figure out how you and your users want to use that information, the business rules that will apply, and how to resolve the inevitable ambiguities that arise.

You learned the importance of interviewing your customers to understand the purpose of your database and the workflow it supports.

You learned the definition of common database terms: entity, attribute, key attribute, and relationships. You learned about one-to-one, one-to-many, and many-to-many relationships.

You tried your hand at creating a data model for a mail, email, and phone list database. With that preparation behind you, you're ready to begin creating physical tables for the entities in your data model.