Data Modeling 101

I keep a small cardboard box on my desk[1]. When UPS delivered it, the box held items purchased from an online store. These days it holds a jumble of invoices paid by clients, paid and unpaid utility bills, insurance documents, a letter from my bank, and other financial documents waiting to be transferred to their long-term home in my filing cabinet.

It's a convenient, but temporary, place to gather unsorted documents in a single container. The difficulty comes when I need a recent document. I don't know if it's in the old cardboard box, or if I've already filed it somewhere in my filing cabinet. If I'm lucky, it's already in the filing cabinet and I can locate it quickly using the filing system I came up with. If it's not in the filing cabinet, I have to dig through everything in the box to find it.

Here's the difference: The cardboard box is *not organized*; it's a box of "stuff". The filing cabinet *is an organized way to manage documents*.

The Cardboard Box Metaphor

You've probably figured out I'm not just talking about a cardboard box and a four-drawer filing cabinet (although they are real enough). They are metaphors for concepts and principles we need to understand in order to build a robust, efficient, flexible relational database application with Microsoft Access.

My box is a precursor to a database. *It is a collection of data about a particular subject with a particular purpose*. In its current state, my "cardboard box database" is only useful as short-term, random storage, so it is not a database. It's not organized in a way that allows me to retrieve things efficiently.

My filing cabinet is more like a real database. It, too, *is a collection of data on a particular subject*. In addition, that information is *organized according to a pre-defined structure, or model*. In database development, we refer to this as a Logical Data Model, or sometimes just Data Model. Once I've processed the documents through that Data Model and stored them accordingly, I can later retrieve them more efficiently.

The filing cabinet holds *information* not *data* because the documents in it are organized and stored according to *a pre-defined system, or model*. Michael Hernandez, who wrote an excellent book called *Database Design for Mere Mortals*, puts it this way.

> *"Information* is data that you process in a manner that makes it meaningful and useful to you when you work with it or view it."[2]

---

[1] I won't hold it against you if you skip this discussion and run ahead to start building your physical 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 information anyway. On the other hand, you don't really have anything to lose by exploring this stuff now, before you get everything all muddled up, do you?

[2] Hernandez, Michael. Database Design for Mere Mortals, Addison Wesley. 2003. Page 45.

I designed the system for my filing cabinet before putting documents in its drawers. That system allows me to add, but more importantly to retrieve, information systematically. It is raw data organized into information so I can be more efficient. And that means it has a Logical Data Model.

It's All About the Data Model

A good way to think about it is that data becomes information when it is organized in a consistent way. Raw data by itself (like the hodge-podge of papers in my cardboard box) is not as useful as that data after it is put into a system.

Moreover, the container you keep the raw data in is less important than the system. If I open a cabinet drawer and shake papers from the box into it at random times, I am no further ahead than if I just keep stacking boxes under my desk. In fact, I don't really need a four-drawer, metal filing cabinet at all. I could organize four cardboard boxes the same way the four drawers in the cabinet are organized because I have a method of organization, or a *Logical Data Model* that determines how the documents are organized.

That Data Model, *or the way you organize the information in which you are interested*, is independent of the *physical container* in which the information will be stored. A properly designed data model can be incorporated into an Access database application, a SQL Server database, or any other Relational Database Management System or application.

Another way to understand this is to recognize that a Data Model is a *logical* concept, not a *physical* object.

Okay, Cardboard Box Guy, Model This

With that introduction, we're ready to look at the process by which you create a Data Model. The way I organized my filing cabinet is simple example. Keep in mind, as you follow the discussion, that I'm talking about the concept that organizes it, not the physical object itself.

What Do We Mean by Data Model?

A data model is a logical description of the things we are interested in and the specific context in which we work. It has three basic aspects.

- ***Entity***
  A statement defining the objects, concepts, and events we want to track.
  It plays the role of a Noun in a natural language.
- ***Attribute***
  A statement identifying a pertinent fact about those objects, concepts, and events.
  It plays the role of an Adjective in a natural language.
- ***Relationship***
  A statement explaining how objects, concepts, and events relate to, or interact with, one another within that context.
  It plays the role of a Verb in a natural language.

Because a data model is not a physical representation of those things, we will not talk about database tables, which are the "physical" objects with which a database works, until we have a logical data model

in place. We're still describing how we'll build the database, not yet building it. When we start building the tables, this logical data model will guide us.

Let's review the organization of the contents in my filing cabinet with a Data Model in mind.

What Things Do I Want to Keep Track of—the Entities

My filing cabinet contains all kinds of documents: auto repair and maintenance records, canceled checks, credit card statements, bank statements, bills from my doctor and dentist, renewal notices from my insurance company along with Declarations Pages for policies, and bills for electricity, water and natural gas. I keep some of them (like tax returns) for legal reasons and others (like insurance documents) for historical reasons. I keep them in a safe place where I can find them quickly if necessary.

These documents are the *entities* — i.e., the objects — around which my filing system is built.

In database terms, an entity is an object, concept, or event in which an organization is interested, and about which it wants to collect and maintain information. Not everything in the world needs to be an entity in a every database, but every entity in any given database needs to be something in the real world which is of interest to the organization.

An entity must be distinguishable from other objects, concepts, or events.

- Entities can be *concrete objects*—like the pay stubs in my filing cabinet.

  You can feel objects with your fingers and see them with your eyes.

- Entities can also be *abstract concepts* like a "forty hour work week".

  You can't see or touch a "forty hour work week", but you know what it is, at least in theory, and you experience it many times every year.

- Entities can also be *events*, such as a sales transaction in which a customer and a vendor exchange a product for money at a specific time. (With online commerce, the location of a transaction is less certain, but the other elements of the event must occur.)

We define entities by stating what kind thing they are and how they are different from similar things of that kind. Let's look at some of the entities in my filing cabinet to see what I mean. The filing cabinet stores DOCUMENTS[3], so the starting point is to pin down what a DOCUMENT is.

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

A DOCUMENT must contain information. We differentiate DOCUMENTS from other PIECES OF PAPER that contain information. Newspapers and magazines, for example, contain information, but they're not DOCUMENTS. DOCUMENTS include only PIECES OF PAPER containing information

[3] In this paper, I adopted the convention of writing the names of entities in all caps. I write relationships in italics. I write differences in standard text.

that must be retained for legal or historical reference. Other PIECES OF PAPER can have information on them, but they're not DOCUMENTS within the meaning of this Data Model.

In another context, represented by a different data model, I would define DOCUMENT differently. I'm only talking about PAPER DOCUMENTS in a metal filing cabinet, not DIGITAL DOCUMENTS on the hard drive of a computer. A different Data Model would augment the PIECE OF PAPER component of the entity definition to account for DIGITAL DOCUMENTS. The media on which the information is stored is the difference, not the purpose for storing it.

Formal Entity Definition

One part of building a Data Model is creating Formal Entity Definitions for the entities in the model. A formal Entity Definition consists of three parts:

- *Term* to be defined (e.g. "a DOCUMENT")
  Any object, event or concept can be the term to define.
- *Class* of things to which it belongs (e.g. "PIECES OF PAPER")
  Classes can also become Terms in their own right in a given Data Model.
- *Differences* between this thing and other members of the same class (e.g. "documents contain information retained for legal or historical reasons").

Generically, that would be stated as:

> ***TERM** is a member of **A CLASS** which is **different** from other members of the **CLASS in this objectively defined way**.*

We recursively define Entities to the level of specificity required by the data model. An Entity defined in one definition can be used as the Class by which another Entity is defined. See Table 3-1.

Entity	Class	Difference(s)
PAPER	is a kind of MATERIAL	consisting of thin sheets of cellulose pulp
PRINTED PAPER	is a kind of PAPER	on which information is printed using mechanical or manual means
DOCUMENT	is a PRINTED PAPER or GROUP OF PAPERS	which is retained indefinitely for legal or historical reference

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

Logical Entities normally become physical tables in your Access database.

Enough With the Entities Already

Obviously, you don't need to go on defining entities and sub-entities indefinitely. It is only necessary to establish enough definitions to fully account for all of the entities of importance in your data model. And if you go up, you reach a level where makes no sense to include the definition in your data model.

For example, while DOCUMENT is an entity in my cardboard box database, PAPER and PRINTED PAPER are not; I manage only DOCUMENTS. MATERIAL, i.e. the product from which PAPER is made, is obviously too general to be of value in this data model. On the other hand, a Data Model

created for a paper manufacturer **would** include MATERIAL and PAPER because they are central to making their products. MATERIALS suitable for making PAPER might not be suitable for making rope, and vice versa. On the other hand, WOOD PULP and COTTON are both materials used in making paper. They belong in the paper manufacturer's Data Model.

In our Data Model, though, we will definitely need to continue defining additional types of documents (i.e., members of the Class DOCUMENT). I track BANK DOCUMENTS, INSURANCE DOCUMENTS, TAX DOCUMENTS, REAL ESTATE DOCUMENTS, etc. because that information is central to the purpose of setting up this database application.

Sorting into Entity Groups

When I pull papers out of my cardboard box to move them into my filing cabinet, I don't immediately transfer them. First, I sort them on my desk: utility bills in one pile, bank records in another, and insurance papers in still another pile. These piles, or DOCUMENT GROUPS, are a way of organizing the raw data.

Later, when I need to find a DOCUMENT in my Filing Cabinet, those DOCUMENT GROUPS help me focus my search. Sorting raw papers into piles is the first step in processing data into information.

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

By the way, that's another point to consider. It really isn't necessary to organize 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.

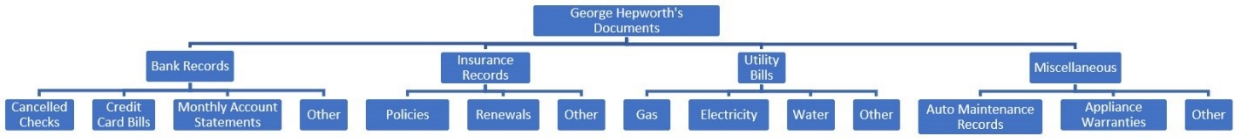

Figure 3-1 Three-level Hierarchy of Document

Figure 3-1 illustrates the three-level *hierarchy* used in my filing cabinet database (with some document groups and document types left out to save space).

- George Hepworth's Documents
- Document Groups
- Document Types

As this model of the data in my Filing Cabinet Database shows, George Hepworth's Documents includes four DOCUMENT GROUPS:

- Bank Records
- Insurance Records
- Utility Bills
- Miscellaneous

Within each DOCUMENT GROUP, there are one or more DOCUMENT TYPES.

For example, the DOCUMENT GROUP called BANK RECORDS includes DOCUMENT TYPES for CANCELED CHECKS, CREDIT CARD BILLS, MONTHLY STATEMENTS, etc.

The DOCUMENT GROUP called UTILITY BILLS includes bills for 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 INSURANCE RECORDS in my filing cabinet comprise another ENTITY GROUP. MISCELLANEOUS is another.

Most Relationships are Not Hierarchical

The DOCUMENTS in my Filing Cabinet Database are organized in a hierarchy, as shown in Figure 3-1. As you can see, entities in a hierarchy fall into sub-groups, each level branching out like the roots of a tree. However, there are other kinds of relationships between entities; one of those is Relational. A Relational Database Management System like Access follows a different method[4]. You'll learn more later in this paper about relational data. For now, the important thing to remember is that 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. I used the Filing Cabinet model as a quick and easy way to get our feet wet with a simple hierarchy.

Organizing Principles—What's More Important?

For this particular data model, a second 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.

Option One: First organize by DOCUMENT GROUP, then by TIME PERIOD

In this data model, DOCUMENT GROUPS would take precedence over the TIME PERIOD to which they apply. Look at Figure 3-2. (I had to reduce the number of levels to get it to fit on the page.)

[4] If you want a good, in-depth discussion of the different ways a database can be organized, I recommend *Database Design for Mere Mortals,* by Michael Hernandez.

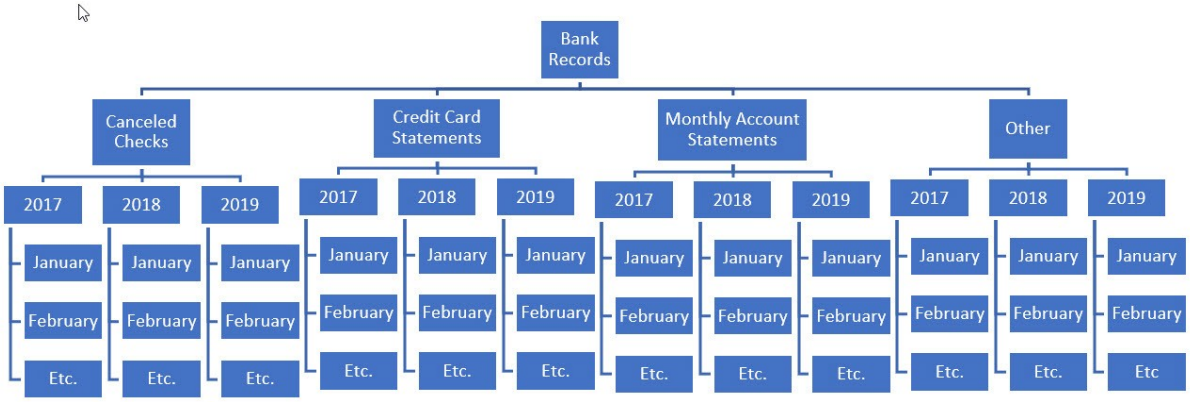

Figure 3-2 Organize by DOCUMENT GROUP then by TIME PERIOD

Under this approach, to find a canceled check for February, 2018, I would first go BANK RECORDS drawer, then the CANCELED CHECKS section, then CANCELED CHECKS for 2018, and then CANCELED CHECKS for February. That's a straight path; however, it requires that I start out knowing exactly how the database is organized hierarchically. That's one of the limitations of a hierarchical database, by the way. You must be familiar with its structure in order to use it effectively.

Option Two: Sort by TIME PERIOD, then by DOCUMENT GROUP

In this data model, TIME PERIOD would take precedence over DOCUMENT GROUPS. Consider Figure 3-3. (Again, it leaves out elements to make it fit on one page.)

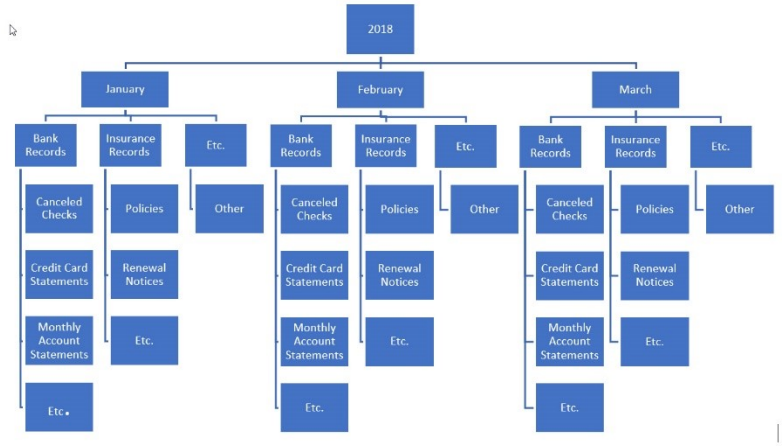

Figure 3-3 Sort by TIME PERIOD by DOCUMENT GROUP

To find a RENEWAL NOTICE for my auto insurance for 2018, I start in the drawer with records for 2018, then the section for JANUARY. In the JANUARY section I find INSURANCE RECORDS and, finally, RENEWAL NOTICES. There may or may not be one in that folder, so I have to back up and check FEBRUARY, or possibly MARCH, and so on until I find what I need. Again, I must the structure of the database use it. If I want DOCUMENTS from 2018, I have to start in the 2018 drawer. In this

second approach, that structure might lead to less efficient searching than the first, but the pattern is the same.

Choosing a Model—Workflow and Business Rules

Even though this is a simple data model, there are at least two ways to organize DOCUMENTS, DOCUMENT GROUPS, and TIME PERIODS. Therefore, we have to choose between data models. Is one of them "right"?

> *As an aside, I've had to address this question when 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. Both approaches do organize things. Both will lead me to the right answer. One, though, is likely to be more efficient, more often.*

Therefore, I have come to believe the most appropriate answer to which data model you should use 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 also means accurately identifying the purpose for which the database will be used and the workflow it will support. Who will use it? How will they use it? What rules will apply to using it? And that, in turn, requires you to spend enough time interviewing your customers to understand the purpose of the database and the workflow it will support. So, before I tell you my choice, let's talk about customers' role in Data Modeling.

Customers are always right—even when they disagree with each other

Your customers include:

- Managers or executives who requested and pay for the database application.
- Subject matter experts who understand the business processes and workflow which it will support and who will have to maintain it after you turn it over to them.
- End users who will use it as part of their regular workload.
- 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[5].

Managers. Managers 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 provides them with a broad view of the organization's long-term goals and plans. Their insight can help you make good decisions about the role your database application will play in those over all plans.

Look to Managers for strategic guidance.

---

[5] Your organization may refer to Stakeholders instead of Customers. I prefer the term "Customer" because it helps me remember that I am providing a product for which someone is paying me. Stakeholders may or may not be paying for it, even though they are definitely interested in the results.

Subject Matter Experts (SMEs). SMEs have technical expertise and knowledge about the business itself and the workflow your database will support. Working closely with them to obtain their knowledge is crucial to success.

Look to SMEs for technical guidance.

End Users. End Users are the folks who have to use your database application in their day-to-day job. If they aren't happy with it, they will avoid using it, up to and including active sabotage. Make sure they are happy. 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 learn from the folks in the trenches, the more likely you are to produce a database application they will use, regardless of what the boss says.

Look to End Users for practical guidance.

External customers. External Customers may be directly or indirectly affected by the database application. Therefore, the degree to which you need to account for their needs will vary. Where there is direct interaction between your client and external customers, you need to do your best to make it as smooth and flexible as you can. For example, if your database application will be used by Customer Service Reps to look up external customer records while speaking with them on the phone, a key factor in providing good service to customers will be making it as easy as possible for your CSRs to find their records. Because it reflects on the organization in the eyes of the external customers, your database application must meet that requirement.

Look to External Customers for a reality check on whether the database application makes sense.

Choosing the Organizing Principle for My Filing Cabinet Data Model

So, with those considerations, I picked the data model I want to use for this task, DOCUMENT GROUP then DOCUMENT then TIME PERIOD, the one in Figure 3-2, not the one in Figure 3-3.

The way I *retrieve* information is more important 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 might 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 insurance coverage, the DOCUMENT GROUP—DOCUMENT—TIME PERIOD approach is more efficient.

For one thing, the homeowners insurance policy renews once each year although 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 (Start with this year then with January, then February, then March, etc.) until I found the folder with that particular DOCUMENT in it. If I haven't yet received one for this year, I start over in the drawer holding last's

years DOCUMENTS and repeat those steps. Even with a good starting guess, the chances are that I'll have to look in more than one place to find it.

For another thing, each homeowners insurance renewal occurs in only one month out of twelve. Once I get to the section of the Filing cabinet where the relevant insurance records are, I'll normally find only a subset of the possible TIME PERIODS there. The folder for the TIME PERIOD in which the homeowners insurance renewal resides may be the only one in that section. That's because I only have to create and store a folder for a given TIME PERIOD if there is something to put in it. There's no point in creating an empty folder for March, 2019 when there are no insurance documents for that month to put in it.

Granted, my Filing Cabinet is not as complicated as the Access database applications 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. I guess you could say this article models the creation of a Data Model.

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

Entities Have Attributes

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

In database terms, *attributes are relevant facts about an entity.* To put it another way, one essential component of a database application is the ability to capture relevant attributes about the entities in the application. I sometimes say that Entities are the nouns of the "Database Language" and Attributes are the adjectives that describe them.

The DOCUMENT entity, for example, has two attributes: what *kind of information* it contains and the *time period* to which it applies. The data model calls these DOCUMENT GROUP and TIME PERIOD. The gas company is not free to send bills for completely unrelated services such as phone lines to my house or for bus tickets.

Furthermore, an instance of an entity has only one value (e.g. March, 2019). A gas company bill can only be for the amount of gas consumed during one period, such as May 15 to June 15, 2019, not for arbitrary, overlapping, or duplicate time periods.

It may seem like I'm only stating the obvious, but when it you create Access tables, you'll need to be able to count on this one-to-one correspondence between attributes and the values they can have.

Ambiguity is a bad thing in a relational database application. Each Attribute must be the smallest, most granular data point you can identify.

Attributes become Fields in a Physical Table

When we create physical tables in our database to store information about logical entities, the relevant attributes will become the fields in those tables.

Here are a few important characteristics of fields.

- Each field in a table contains one attribute, and only one.
- Each attribute is stored in one field, and only one.
- Therefore, two or more fields can't contain the same attribute.

What do I mean by this? I mean that you can't put two kinds of attributes in one field. For example, if you have an attribute for "Individual Name", you can end up with as many as four different data points in it. "George" is a first name; "Russell" is a middle name; "Hepworth" is a family name; "Sr" is a suffix. Both "George" and "Russell" can also be last names, and they can be arranged as "Russell George Hepworth". In fact, recognizing that they are interchangeable is a really strong signal they are not supposed to be in one field together. Further, if "George" is a first name for one person, and a last name for someone else, they need to be assigned to fields in a physical table accordingly. I'll return to this point again later in this paper.

Enough is Enough When it Comes to Attributes

Obviously, we are not interested in capturing every possible fact about an entity, just as we weren't interested in every possible entity. For example, I don't care whether my bank statements come on blue paper or pink paper, that they are printed on one or both sides of a page, or that bills from the phone company list international calls separately from other long distance calls. For this database application, I don't need to keep track of those irrelevant attributes. A different organization, such as an office handling international trade, might track international calls separately from domestic calls.

However, the success of your database application depends on identifying all of the attributes for your logical Data Model that will be of interest to you before you start creating physical tables. It's easy to alter a blueprint before starting construction, but moving a concrete wall in a decade old building, not so much.

Entities, Attributes and Relationships

So far, my discussion of entities, attributes, and data models has been relatively informal because I want to help novices among us grasp the basic concepts before tackling the industry terminology that goes with them. I have over-simplified some concepts and omitted others. Let's correct that deficiency.

What follows is a more formal review of what we've covered so far, and an expansion to cover the third component, relationships. We'll need to use examples that are slightly more complex than my hierarchical cardboard box filing system. It's served its purpose in getting us started, but it will now have to be left behind.

Entity

An entity can be:

- an object
- a concept
- an event.

In other words, an entity can be a physical thing, it can be an idea or formulation, or it can be something that happens.

As I mentioned earlier, I think of Entities as the Nouns in a Database Language.

Physical Entities

The personal computer sitting on the floor under my desk, a Lenovo with Serial Number 07130-2M2-0120, is a physical object, i.e. it's one instance of one physical entity: Personal Computers. It's not a typewriter, not a coffee pot and not a chair. Those are other physical entities.

The Serial Number on my PC uniquely identifies it as one particular PC in the universe of all other PCs. It has other attributes, of course, like the manufacturer, model, motherboard, hard drive, and so on. But one can plug it into a power outlet, press the power button and use it to do work. It is physical entity called a Personal Computer.

Conceptual Entities

An entity can also be an abstract concept like a contract. While a contract or transaction can be memorialized in writing, that's not a legal requirement for the contract or transaction to exist. A verbal contract is binding, even though its existence might be harder to prove without a written document. There's even an old saying about that. "My word is as good as my bond." No need to write it down; it's a contract when I agree it's a contract.

Event Entities

Entities can also be events such as sales transactions. The transaction event in which I purchased the PC from the local computer store occurred on October 21, 2017. That sales transaction included a salesperson at the computer store, the hardware, my credit card, and me. The purchase itself is an event entity. The purchase date, the PC, the salesperson, and myself are all attributes of the purchase event, and even though I no longer have the cash register receipt for the transaction, it still exists.

Entity Sets

An Entity Set is a group, or set, of entities of the same type. The canceled checks from my checking account are an Entity Set. They are all one kind of thing, but are unique and distinguishable from each other in several ways, such as the amount, payable to, paid date, and so on.

Another example of an Entity Set would be my laptop and desktop computer. They belong to the same Entity Set of personal computing devices.

Common Members Between Entity Sets

Two or more entity sets can have members in common. For example, the Entity Set *school employee*, which consists of everyone who worked at my daughter's school, can have members in common with

the entity set *school parent*, which consisted of parents of students in the school. Some school employees also enrolled their own children in the school. The database term for this principle is that entity sets *need not be disjoint*. We need to be alert for such cases because it can impact how our Data Model works. Suffice it to say at this point, that we need to be careful about building too many physical tables for such situations.

When you are deciding which Access tables you'll need to represent entities, you'll learn more about the importance of this principle, particularly with regard to deciding how to think about what an Entity really is, and where to attach attributes to it.

Attributes

Each entity has one or more attributes. Attributes fill the role of Adjectives in a Database Language.

For most entities, such as school parent, there is a fairly well-defined set of relevant attributes: for example, first and last names, street, city, state and zip, email address, home phone, and work phone. School employees have the same attributes. They differ only by the Role they play in the lives of students at the school.

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 *male* or *female*, that attribute may or may not be in a database of school parents. The same is true of school employees. However, is needed for students. We need to know if a student is a boy or a girl.

You don't know whether you need to track any given attribute, of course, until you've interviewed the customers who will use the database application. If there is a business reason to keep that information, you'll need to include it. Otherwise, leave it out.

In my experience, the larger problem in most development projects has been failing to identify and include all of the relevant attributes in the initial data model. It's better to start with a list of all possible attributes and whittle it down as you refine the data model, 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 application.

Domain

Domain refers to the permitted values for an attribute. For example, the Domain of GENDER has two members, male and female[6]. For ZIP CODE, the Domain of valid values is strings of either five or nine positive integers, with no letters or other characters allowed. For POSTAL CODE in Canada, though, the Domain of permitted values does include letters and different number of digits. So, once again, Business Rules overlap with data requirements.

---

[6] When I first wrote the book on which this article is based, I had no idea how fluid gender would become in the following decade. The Gender domain has expanded well beyond the binary choices stated here. But recognizing that fact is also highly relevant to understanding how a good table design needs to be established. Remember, each attribute needs to end up in one field in a physical table. Back then, I had only two choices to account for, but as time moves on the list grows longer. The good news is that a properly designed data model makes that evolution relatively easy to handle, at least at the technical level. So, even though I started with two values, my database application allows me to add more easily.

Attribute/Data Value Pairs

For each entity in an entity set, there are one or more pairs of attributes and data values. What that means is that, 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. Let's see if we can tease some meaning out of that sentence.)

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

For example, consider a shirt on sale at the local thrift shop. 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 a shirt's color without specifying what that color is: red, blue, pink, etc. Red, blue, and pink are some of the possible data values for COLOR for this particular entity, SHIRT. And you can't say that a single SHIRT is simultaneously pink and blue, unless of course you mean it has stripes of both colors. But you can't tell the sales person you want "a colored shirt" without adding it needs to be pink color.

Obviously, cotton, silk or polyester are not possible data values for COLOR, although they would be appropriate for the MATERIAL attribute of SHIRTs. Each attribute has its own set of values. And you can assign only one of those to each instance of the entity.

Later, when you learn about the more formal rules of normalization, you'll get another look at the importance of this point. For now, just remember each attribute consists of an attribute and its corresponding data value, taken from a domain of permissible values for that attribute.

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

Attribute	***Data Value***
First Name	William
Middle Name	Henry
Last Name	Carterfield
Nick Name	Bill
Birthdate	08/10/1980
Inactive Date	6/30/2001
Relationship Type	School Alumnus

Table 3-2 Attribute-Data Value Pairs

As Table 3-2 shows, each attribute for one *individual* has one, and only one, corresponding data value. For this person, First Name is William, Nick Name is Bill, and so on.

We don't store both William and Bill in the same attribute.

Just in case you're wondering about it, that includes things like "William (Bill) " for a first name. Each attribute has one and only one data value in a relational database: "William" is a First Name, "Bill" is a Nick Name. "William (Bill)" is neither and is not allowed.

Relationships

Up to now, I've used the terms relationship and relational in less formal ways. I briefly talked about the hierarchy of DOCUMENTS in the course of discussing how we came to group documents for filing. A hierarchy is one type of organization, but not the type of relationship in a Relational Database Application you will build with Access. So let's see how that works next.

Relationship

A relationship is a formal association between entities in different entity sets[7].

I sometimes call them the verbs of the Database Language.

For example, consider HOUSEHOLD and INDIVIDUAL, which are entity sets in a database of school families. This database application keeps track of households associated with the school and the individuals associated with those households. It also tracks how students are associated with the school.

In that database, an INDIVIDUAL is any person in whom the school is interested: a student, a school employee, or a parent or other relative of a student in the school. A HOUSEHOLD is defined as one or more INDIVIDUALS who share a single physical address[8].

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. We don't require them to be family.

Initially, one might assume that an INDIVIDUAL can only reside in a single HOUSEHOLD. However, we need to allow for cases when someone spends time in multiple households. That could occur, for example, when parents divorce and share custody of their children.

Role-Based Relationships

This part of the school database application is a role-based relationship. We define the *relationship* between HOUSEHOLD and INDIVIDUAL by specifying the *role* each person plays in the relationship

---

[7] Although a full discussion is beyond the scope of this paper, the term "relation" in Relational Database comes from the mathematics on which the theory of Relational Databases is based, and it does not really refer to Relationships between entities, which is a common misunderstanding. It's a happy coincidence that the same word appears in both.

[8] It's not critical to this discussion, but it is interesting to recognize that this definition includes households consisting of a married couple (with or without children), a single parent or person, unmarried partners, a parent and grandparent, and other permutations involving at least one person.

between them. For the purpose of this relationship in this context, the role of the INDIVIDUAL is *resides in*.

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

Contrast that to someone who comes to the location once a week to do house-keeping. We're tracking residents of a household, not employees of that household.

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

Contrast that to a PLACE OF WORSIP, which is a location INDIVIDUALS regularly visit, but do not reside in.

As we just saw, in a 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 relationship.

A CUSTOMER *purchases* a PRODUCT.

A PRODUCT *is purchased by* a CUSTOMER.

A STUDENT *enrolls in* a MATH CLASS.

A MATH CLASS *consists of* one or more STUDENTS.

The way you phrase the relationship may vary. If you're primarily interested in a catalog of current classes for a report, and secondarily interested in who enrolled in each, you'll probably phrase the relationship the second way. If you're creating a roster of students with their classes, it would be the opposite. As long as you've pinned that relationship down, it doesn't matter how you phrase the description.

Descriptive, or Status, Relationships

Some relationships are statuses, 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, i.e. their status on that date is present, tardy or absent. ATTENDANCE RECORD captures that by specifying the date and attendance status of each STUDENT on that date.

Relationship Set

A relationship set is a group 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 is:

A CUSTOMER *purchases* a PRODUCT. The relationship set, therefore, would be the multiple customers and the products they each purchase.

For example, we have CUSTOMERS named John, June, and Jerry. We have four PRODUCTS—a red bicycle, a blue bicycle, and two green bicycles. The RELATIONSHIP set between them could be these:

- On Monday, John purchases the red bicycle.
- On Tuesday, June purchases the blue bicycle.
- On Tuesday, Jerry purchases one of the two green bicycles.

Although the other green bicycle is still in the PRODUCT group, it won't participate in a relationship with a CUSTOMER until the customer purchases it. The second green bicycles is not a member of that purchase relationship set until it is purchased by someone.

This is important because our relational database application has to capture the relevant attributes of that relationship set in a physical table designed for that purpose. It must allow a PRODUCT to exist without a CUSTOMER.

Relationship Types

We are primarily interested in three types of relationships:

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

One-to-One Relationships vs Many-to-Many Relationship

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. In a many-to-many relationship, each entity in one entity set is related to one or more entities in a second entity set, and each entity in that second entity set is related to one or more entities in the first. What does that mean in the real world? A good example is Company Cars for Employees, which can work two ways.

Business Rules for Company Cars

Many corporations provide company cars for employees. The business rules of the company determine how cars are allocated for the use of employees. The appropriate relationship between cars and employees is determined by those business rules.

The company to do this two different ways, depending on which business rule they adopt.

One employee gets one car—One to One Relationships

Company GRH provides each sales manager with a car for that manager's use for a specific period of time. Sales managers drive only their own cars. 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 would apply). This relationship is one-to-one:

One SALES MANAGER *is assigned to* one COMPANY CAR.

One COMPANY CAR *is assigned to* one SALES MANAGER.

The data model to manage this company's company car fleet needs to define a *one-to-one relationship* between SALES MANAGERS and COMPANY CARS.

One-to-one relationships are the least common. Moreover, many one-to-one relationships can be reduced into one table in a physical database. Pay close attention to these relationships in your data model to be sure you set up your tables correctly. If you see them, stop and review the situation to be sure that's what you really have and should have.

Any employee can use any car—Many-to-Many Relationships

Company XYZ has a fleet of cars that employees check out as needed. Each car can be used by multiple employees on different days and each employee can use multiple cars at different times. This business rule leads to a many-to-many relationship in a database application. The relationship is:

An EMPLOYEE *is allowed to use* one or more COMPANY CARS.

A COMPANY CAR *is assigned to* one or more EMPLOYEES.

Let's save a more detailed discussion of Many-to-Many relationships for the longer discussion below. First, let's review the third, and probably most common type of relationship, One-to-Many.

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, potato growers cultivate *one or more* fields of potatoes. This is a one-to-many relationship. One grower has many fields. One field belongs to one grower. The data model for a government agency tracking pest control for local potato farmers 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 often the easiest to define and set up in a database. Here are some other examples of one-to-many relationships I've found in database applications I've built for customers.

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

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

An elementary school CLASS *enrolls* one or more elementary school STUDENTS.

An elementary school STUDENT *is enrolled* in one elementary school CLASS.

In the one-to-many example of a STUDENT in an elementary school 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 for the school year 2018-2019. The seventh grade includes Josefina and 22 other students: one class, many students. All 23 student attend that seventh grade class.

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 other classes the college offers, so each student has a unique schedule of classes.

For example, John O'Hara has Math, English, and Biology classes while Keiko Yamasaki has Math, Botany, and Electrical Engineering classes. This is a Many-to-Many relationship, where each student participates in one or more classes and each class enrolls one or more students.

Let's look at Many-to-Many relationships in more detail.

Many-to-Many Relationships

In a many-to-many relationship, each entity in one entity set (students) can be related to one or more entities in a second entity set. These relationships can occur sequentially or simultaneously. Any entity in the second entity set (classes) can be related to one or more entities in the first entity set (students) at some point in time.

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.

Another business rule 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, first come-first served. Many cars can be assigned to one employee over time. Many employees can be assigned to one car at *different* times. That is a *sequential* Many-to-Many relationship.

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

A CLASS *enrolls* one or more STUDENTS.

A STUDENT *enrolls in* one or more CLASSES.

Because one or more of these enrollments can occur at the same time, this is a *simultaneous* Many-to-Many relationship.

By the way, here is another many-to-many relationship I encountered while building a database application for an elementary school. This relationship can be either *simultaneous* or *sequential*, or *both*.

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 frequently happens. 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 caring for one or more children. It can be an unmarried couple and one or more children. It could be a single parent with one or more children and one of their grandparents. It can even be a single person working at the school.

Many students don't live exclusively in one HOUSEHOLD. They spend time in divorced Mom's house and time in divorced Dad's house. Or they live with grandparents and visit parents in another household. During one school year the student lives in one Household, but lives in a different Household the next school year. The attendance database application I built for this school, therefore, needed to accommodate both *simultaneous* and *sequential* many-to-many relationships between STUDENTS and HOUSEHOLDS, based on the real world circumstances the interview phase of our data modeling process revealed.

Business Rules and Data Models

Each organization has 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 need to know not just what entities are involved, but how your organization applies its business rules to those entities.

In almost every situation where you define relationships between entities in your model of the data your database will track, you'll have to consult 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 relational database application with the appropriate relationships defined.

Business-Specific Entity Definitions

The community college example needs to accommodate multiple students in multiple classes, but it shouldn't allow students to sign up for more than one class during the same time period on the same day(s) during the same enrollment period. To achieve this, the college must have a different definition of CLASS than the elementary school, one that includes week days and times as well as enrollment periods. Another way to look at this would be to say that calling something a "CLASS" doesn't tell you everything you need to know about it in either the community college environment or the elementary school environment. Just because you think you know what a term means doesn't imply someone else in a different environment will see it the same way you do. Ask, define, communicate, clarify.

Key Attributes or Unique Identifiers

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

- Identifying the entities of interest
- Identifying their relevant attributes
- Identifying the relationships that link entities with one another

To complete the process for a relational database application, we need one additional piece of information. That is the *key attribute* that uniquely identifies each entity in each entity set.

It Quacks Like a Duck, and Not Just Any Duck

Think of a bucket of water with a dozen identical yellow rubber ducks floating on top. The ducks slosh around; you can't count on any one of them being in any particular location in the bucket at any particular time and you can't tell them apart by looking at them. If you pull one out, then put it back in the bucket, you may have trouble finding that one again because there's nothing to mark it out from the others. Fine for a kid's game, perhaps, but not good if you need to retrieve that one particular duck later.

On the other hand, when you pluck a duck out of the bucket, you can number it with a marker. Then, when you put it back in the bucket, you uniquely identify it with that number on its back. If you number them before you even put them in the bucket, your task is simpler still. However, as soon as you put marker to duck back, you no longer have identical yellow rubber ducks. They're now all unique because of that unique identifier on their backs. Even if you use the digit "1" for a duck, and the word "one" for another, they're still distinguishable. They're now unique. And that's a good thing when it comes to finding the "right" duck in the bucket.

Key Attributes

This is where Key Attributes in a Data Model come into play.

In a relational database application, each member of an entity set has a unique identifier so that you can tell one member from another. Most entities have multiple attributes. One of them, or sometimes a combination of them, will be the Key Attribute(s) that uniquely identify each instance of the entity from all other instances of it. The number on the duck's back is unique, even though they are all yellow, all the same size, the same buoyancy and shape, and all made of rubber. They now have a Key Attribute—a number on their backs. If you had a pink duck, a blue duck, a pink fish and a blue fish, both attributes (color and species) are needed to uniquely identify each. That is, a composite of two attributes is the Key.

A key attribute is either one of many naturally occurring attributes of an entity or a composite of two or more attributes. The important thing is that, the key attribute has to be a unique value for each member of that entity set.

In my office, the serial number on the back of my Lenovo PC, 07130-2M2-0120 is an example of a Key Attribute. It is unique within the production of that model; no other PC produced by Lenovo has that identical serial number. The PC under my desk has other attributes, such as CPU speed, RAM, hard drive storage capacity, USB ports, and so on. However, only the serial number can uniquely differentiate this one from all other PCs with similar attributes produced by Lenovo.

Two different manufacturers could issue the same serial numbers. It's unlikely but not impossible. So even though I consider Serial Number a good candidate for the key attribute, In the back of my mind, I can't ignore the possibility it might not be universally unique, especially if I had to track thousands of personal computers in an organization.

Perhaps people's names are a better example of this problem with unique identifiers.

I can be uniquely identified by referring to a combination of three or more attributes. My three names, "George", "", and "Hepworth", are not individually unique to me. There are many "Georges", many "Russells", and many "Hepworths" in the world. I also have one cousin and one nephew with whom I share two of the three names, albeit in different combinations. My name is probably not even unique in that combination. But that's only the people I know or have heard about. It's possible that someone I've never met is also named "George Russell Hepworth". This is an important consideration because it means you cannot assume that what you think might be a Key Attribute is actually going to stand up to real world experience. How do we deal with that? Perhaps we can add an additional attribute more likely to be unique.

Social security number is sometimes offered as a unique identifier, especially when combined with a name. In a logical data model, we might consider the *combination* of first name, middle name, last name and SSN enough to uniquely define a person. We won't go there for other reasons, though. Principally, that's because SSN isn't really universally unique, and privacy concerns mean we shouldn't use it anyway. Many organizations deal with this problem by creating their own unique Identification or Account Numbers. That allows them to hire John Allen Partridge from Muskogee, Oklahoma and John Allen Partridge from Oakland, California. If they didn't use an Employee ID, they'd have to fire the guy from California in order to hire the one from Oklahoma.

Getting back to personal computers, these two key attributes (first, last and middle names for the person and Serial Number for the computer) uniquely identify the relationship between any two entities, PERSON and COMPUTER EQUIPMENT, in my HOUSEHOLD.

PERSON (KEY)	**Owns**	**PC Equipment**	**Serial Number (KEY)**
George Russell Hepworth	owns	LENOVA	07130-2M2-0120

Table 3-3 Relationship Example with Key Attributes (Small Domain)

In a corporate environment, we would add a unique identifier to the PERSON name so we can be sure we're talking about a person in Sales, as opposed to a person with the same name in the IT department.

PERSON	**Employee ID (KEY)**	**Uses**	**PC Equipment**	**Serial Number (KEY)**
George Russell Hepworth	GH10101010	uses	LENOVA	07130-2M2-0120

Table 3-4 Relationship Example with Key Attributes (Large Domain)

Take note of the fact that creating a surrogate value (Employee ID) to serve as the Key Attribute means we no longer have to consider the Person name to be unique. We can hire, every "Diane Raquel Griffin" in the whole state if we want to. As long as each one gets their own Employee ID, we're all good.

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.

Most of us will seldom, if ever, use natural Key Attributes as the Primary Key in physical tables[9]. Consider, for example, the problem we've identified with names and social security numbers. In our logical data model, it the combination of all four attributes uniquely identifies a person. Access would allow you to set them up as a composite key in a table, but working with all four attributes all of the time would lead to complications you'll want to avoid as much as possible. A single, Surrogate Key is usually handier.

SSN by itself appears attractive as a Key Attribute and, in some contexts, it may start as a Key Attribute in a *logical* model. In reality, it doesn't work as the Key Attribute in a physical table. In fact, there are good arguments against even storing SSN in most databases, primarily because of privacy concerns. In a back-handed way, I only brought it up in order to caution you against considering it.

Further Study

The foregoing discussion of entities, attributes, key attributes, and relationships is really no more than an introduction to a fascinating area of study. Still, I want you to be comfortable with some of the concepts of entity, attribute, and relationship, so you'll be ready to start building well-designed physical Access tables.

Creating an Informal Data Model

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

Your First Data Model

Start by listing all of the things you think you'll need to know for this kind of database application, things like names, email addresses, mailing addresses, and phone numbers. I also want to track significant milestone dates, so I include birthdates and the dates on which the individual is no longer active in the family. Don't look at my list until you've written down as many as you can think of yourself. List your entities and attributes, and select the key attributes (the ones that uniquely identify each entity. For now, don't worry about whether they are natural or surrogate keys.

---

[9] The discussion of "Natural Keys" and "Surrogate Keys" usually arouses controversy. It's beyond the scope of our current discussion, unfortunately. The natural key would be my three names, the names found in the real world. The surrogate key would be the employee ID assigned by an employer. I am in the "Surrogate Key" camp, i.e. I would use the employee ID, not the natural key names in a table. However, the other camp, which prefers "Natural Keys", has equally valid arguments for that choice. If this topic interests you, it's well worth the time to research it.

Also, write descriptions of entities and their attributes. Then see if you can sketch out the relationships between entities. Here are some simple conventions to get you started.

1. Enclose the names of entities in rectangles.

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

3. Relationships can be one-to-one, one-to-many, or many-to-many.

   Indicate relationships between entities by using "1" for the one side and the symbol "∞" for the many side. In many-to-many relationships use two "∞" symbols. In a one-to-one relationship, use two "1" symbols.

   Because each person can have one or more phone numbers, we place the ∞ symbol next to phone number entity. Because one or more persons can use each phone number, we place the ∞ symbol next to person as well. This designates it as a many-to-many relationship.

My Data Model

Let's look at my Logical Data Model for a mailing address, email and phone number database application. While I explain what I did and why, compare that to your model and the choices you made. The following discussion is fairly detailed at some points, probably more detailed than would otherwise be justified for a database as simple as a mailing, phone and email database application. That's so we can be sure to highlight the thinking ***behind*** the choices. At the same time, I'm going to skip over some of the more technical aspects of data modeling for now in favor of what I think are more practical ways to approach the material.

My intent *is not* to make you a fully qualified data modeler or database developer. It *is* to illustrate ideas and principles I feel are important to grasp before you create physical tables in a 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 here. You'll have to justify other things based on the Business Rules you want to follow. Since this is possibly your first attempt at creating a logical data model, don't worry if you didn't get it quite right on the first try. You'll get better with practice.

I defined the purpose of this database to be mailing, email, and phone numbers *along with birthdays and inactive dates*. 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.

In my mailing and phone list data model for family and friends, I am interested in the entities and key attributes shown in Table 3-5. Table 3-5 also includes definitions for the entities and an example of each.

Proposed Entities and their Key Attributes

Entity	Definition	Key Attribute(s)	Example
Individual	Person for whom I record a phone number, address or email address, and a household	First, Middle and Last Names	George Russell Hepworth
Household	Group of people who reside at a single location	Household Name	George Russell Hepworth household
Household Role	Roles in a Household held by members	Household Role	Head of Household
Individual—Household	Individual(s) and the Household(s) to which they belong	First, Middle and Last Names (from Individual) Household Name (from Household)	George Russell Hepworth is a member of the George Russell Hepworth household
Address	Designation for the physical location where Households reside	Street address, City, State, Country	10101 NE Main Street, Afton, Wyoming, USA
Household—Address	Household(s) and the Address(es) where they reside	Household Name (from Household) Street address, City, State, Country (from Address)	George Russell Hepworth household resides at 111 Cloud Ave, Newcastle, Washington, USA
Country	Countries where Household are located	Country Name	USA
Email Address	Identifies an email destination to which email messages are delivered.	Email with domain	NormalizeThis@myemail.com
Individual—Email Address	Individual(s) and the email address(es) they use	First, Middle and Last Names (from Individual) Email with domain (from Email Address)	George Russell Hepworth gets spam at NormalizeThis@myemail.com
Phone Number	String of specific digits one phone uses to connect to a second phone	Country Code, Area Code, exchange, number, extension	01-555-555-5555
Individual—Phone	Phone numbers used by Individuals	First, Middle and Last Names (from Individual) Country Code, Area Code, exchange, number, extension (from Phone Number)	George Russell Hepworth answers calls to 01-555-555-5555

Table 3-5 Entities and Key Attributes

This is the minimum set of logical entities I need for a family contact database application. I would possibly include additional *tables* in a physical database application built to implement this data model.

Proposed Entities, Key Attributes, and Additional Relevant Attributes in a Logical Data Model

In addition to the key attributes in Table 3-5, I am interested in non-key attributes, such as nicknames and any suffixes (e.g., Jr for sons, Previously Known As for married women), birth dates and deceased dates of people, 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.) Table 3-6 includes Key and Non-Key Attributes.

Entity	Relevant Attributes for this Entity
Individual	First Name (Key for Individual)
	Middle Name (Key for Individual)
	Last Name (Key for Individual)
	Nickname
	Suffix (Key for Individual)— e.g. Jr., Sr.
	AKA or Previously Known as—maiden name, pre-adopted name
	Birthdate
	Active Date — individual marries into the family, is adopted, etc. May be the same as a Birthdate
	Inactive Date — normally, death date, but persons may become inactive for other reasons
Household	Household Name (Key for Household)
	Active Date—Household created by marriage, children moving away from home, etc.
	Inactive Date—Household dissolved by death, divorce, etc.
Individual — Household	First Name (Key from Individual(s))
	Middle Name (Key for Individual(s))
	Last Name (Key from Individual)
	Suffix (Key from Individual(s))
	Household Name (Key from Household(s))
	Active Date—Date on which this individual joined this household, e.g. birth, marriage, adoption
	Inactive Date—Date on which this individual left this household, e.g. death, divorce
	Household Role (Key from Household Role(s))
Country	Country Name (Key for Country)
Address	Street number and name (Key for Address)
	Second street number and name, e.g. Apartment Number (Key for Address)
	City (Key for Address)
	State (Key for Address)
	Postal Code (Key for Address)
	Address Priority—Principle, Business, Summer Home, etc.
	Country Name (Key from Country)
	Address Type—Street, P O Box
Household—Address	Household Name (key from Individual)
	Active Date—Date on which this household initially occupied this Address
	Inactive Date—Date on which this household left this Address
	Street number and name (Key from Address)
	Second street number and name, e.g. Apartment Number (Key from Address)
	City (Key from Address)
	State (Key from Address)
	Postal Code (Key from Address)
	Priority — Primary, Secondary, Inactive
Household Role	Household Role (Key for Household)
Email Address	Email address with Domain (Key for Email Address)

Individual—Email Address	Email address with Domain (Key from Email Address)
	Email Type—work, personal
	First Name (Key from Individual)
	Middle Name (Key from Individual)
	Last Name (Key from Individual)
	Suffix (Key from Individual)
	Priority — Primary, Secondary, Inactive
Phone Number	Country Code, Area Code, exchange, number, extension (Key from Phone)[10]
Individual—Phone Number	First Name (Key from Individual)
	Phone Type—Work, Personal, Mobile
	Middle Name (Key from Individual)
	Last Name (Key from Individual)
	Suffix (Key from Individual)
	Country Code, Area Code, exchange, number, extension (Key from Phone)
	Priority — Primary, Secondary, Inactive

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

Here are the relationships needed.

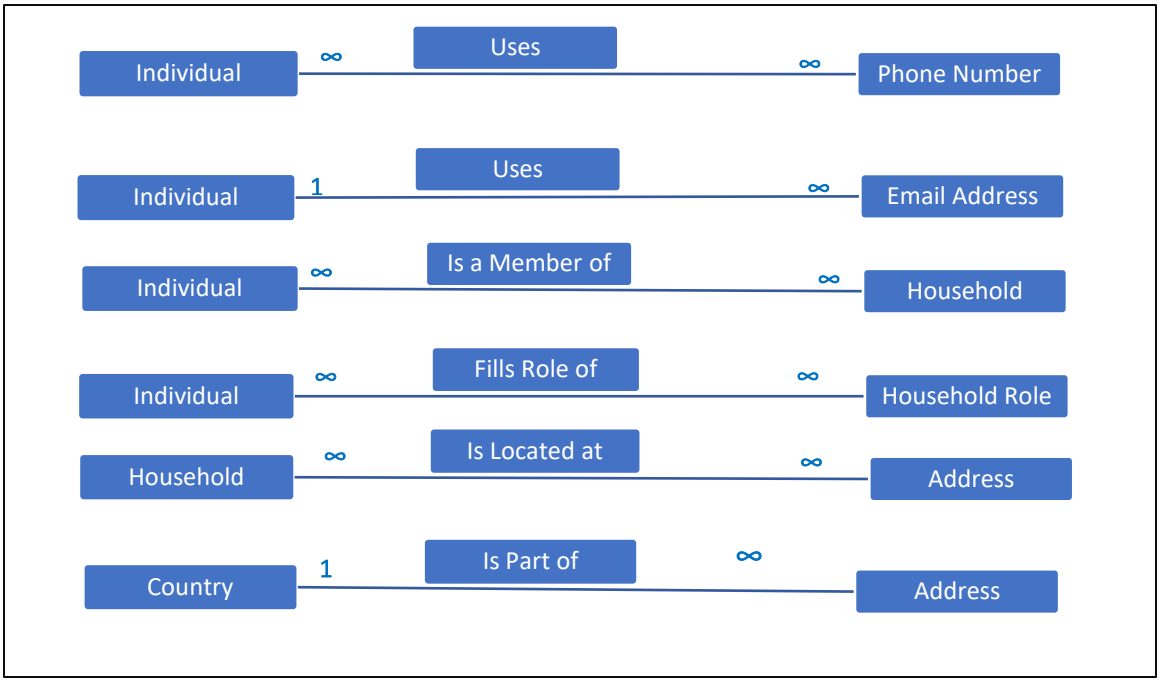

Figure 3-5 Logical Data Model —Relationships between Entities

Proposed Entities and Relationships

The Country attribute allows me to filter out US addresses from international addresses in case I need to target specific groups. The household name is useful when I want to send a Christmas card to all members of a household. Identifying people and the roles they play (head of household, spouse,

[10] Both Phone Number and Address can be further analyzed and broken down into smaller attributes depending on the purpose of the database application. For the purposes of a small, family-oriented application basically intended to work like an electronic contact book, the way it's done here should be fine.

members) may be useful for other reasons. We'll see when we start adding data and using the database application.

Hedging my Bets

I 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 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 the Data Model until I make up my mind about it.

An attribute that may prove to be irrelevant is the "type of" for phone numbers, email and addresses. I think I might need to know if an email is work or personal, but that may turn out not to be useful. So, I'll add it now with the proviso I may decide to drop it sometime. It's easier to pull it out than to put it in later.

Managing Data—Delete vs. Inactive

I also decided to include the date when a family member is inactive (which usually means 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. Individuals, phones, email addresses, households and address records also have an Inactive Date.

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. That allows me to display a list of August birthdays, for example, and quickly look up the appropriate address or addresses at the same time so I can send cards[11]. However, 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 interested in memorializing divorce dates in my contact database application, so I didn't include that information in my model per se for Individuals (A different database application might be the opposite). I can see making them inactive in one household, but they didn't fall off the face of the earth after the divorce and I don't want to consider them inactive as people because I care about them, so I can't delete them from the database application.

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.

---

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

The principles of good database design tell me that I don't remove records from a database.

One key principle in developing any database application is that a database is good at keeping both historical and 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 deceased people, then, I want to record their status. That way, deceased date (or more broadly inactive date) prevents me from accidentally including them on mailing lists for birthday and Christmas cards, etc.

Status

By the way, deceased date is a specific example of a generally useful piece of information found in nearly all database applications: *Active and Inactive Status*. Active Status is, in turn, an instance of a set of Statuses that usually appear. Other statuses might be "Include on Mailing Lists", for example, so you can avoid sending a birthday card to that curmudgeonly uncle who complains.

The general idea is that, anytime you have an entity whose status can change over time, you can often best handle it by recording the date(s) that status changes rather than deleting or adding new records.

Who's Related to Whom

Although this is not the only way to do it, I decided *addresses* belong to households while *phone numbers* and *email addresses* belong to individuals. This isn't an obsession with the specifics of addresses, phones and email addresses. Rather, they provide a simple basis for discussing the thinking involved in establishing a logical Data Model.

If I ask someone 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. Addresses are more closely aligned with households, not individuals.

However, that isn't true if I ask for a phone number. They'll often respond with their own question, "Do you want my cell phone, my work phone, or my land line?"

The same is true 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. Granted, an organization might have a generic inbox for inquiries or customer service, but no one person actually uses that one personally.

So, when I set up this database application, I decided I would relate mailing addresses to households and email addresses and phone numbers to people. That makes sense within the context in which the decision applies. Other Business Rules might lead to a different decision.

Gray Areas, Trade-Offs, and Lessons Learned

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

Suppose that some of the people in my database application are actually business associates rather than family. The only address I might have for them would be a work location; I don't know their home address. Moreover, I don't know any of the other household members at that person's address. In that sense then, addresses could be seen as being related to the person rather than the household.

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

In both cases, the 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. I only need one address per household to do that. A different database application might end up with a different implementation of the data model.

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 matters. During the workday, I call the person's work or cell number or use their work email, but during the evening or on the weekend, I 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 household (although those of you who have teen-agers may feel otherwise[12]) and each of those family members uses more than one phone.

Other phones are used almost exclusively by one person. For example, nearly all cell phones are personal phones. There is only one person in that relationship with that phone. It's no longer rare, in fact, for a household not to have land lines. 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, by the way, not because I'm 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 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 creating, evaluating, and revising your data model in light of the definitions and business rules results in a better understanding of how the relationships work. You will be better able to make good decisions about the potential advantages and disadvantages of each approach. Doing so will help you head off problems down the road.

---

[12] This statement is already outdated, isn't it? I'm leaving it in on the off chance some of you may still not own a smart phone.

Trade-Offs in an Interface

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

Summary

In this discussion, I've given you an overview of the initial phase of creating a new Access database application, starting with the cardboard box-to-filing cabinet metaphor in which raw data is processed into information.

- You learned that raw data becomes information when it is organized according a logical 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, which business rules will apply, and how to resolve the inevitable ambiguities that arise.
- You learned the importance of interviewing customers to understand the purpose of a 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 logical 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. Good luck.