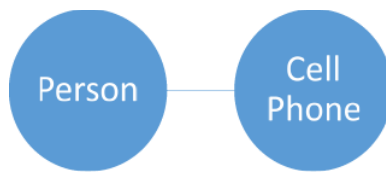
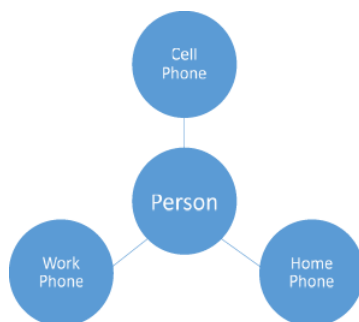


4.2: Data Relationships

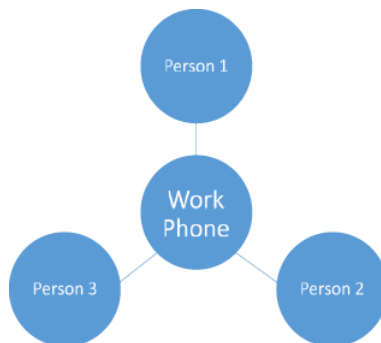
Before we begin to design our database, we need to understand the different relationships that can exist between two pieces of information. For this example, we will start with two imaginary tables. The first will be names, and the second will be phone numbers. If we first consider a person with a cell phone that has no other numbers they can be reached at, we see a **one-to-one** relationship—the phone number for that cell phone is the only one associated with that person, and that person is the only one you reach when you call that number.



This does not cover all phone uses, though. Many of us still have phones at home or at work that are used by multiple people. In this case, the relationship between that phone and its users is **one-to-many**, as more than one person can be reached by using that phone number.



In reality, both of these are probably true for most people. This means that one number can represent many people (calling a house or business) and one person can be reached via multiple phone numbers. In this case, we have a **many-to-many** relationship where multiple values of the same table can relate to multiple values of another table. In this example, of all numbers (work, home, or cell) are stored in the same table, there can be multiple values connected to either side of a given number.



When we apply the theory of normalization to the database we are about to design, it is important to keep these relationships in mind as it indicates how we should structure our database. A one-to-one relationship can be resolved by keeping both pieces of information in the same table or by including a reference in either of the two tables to the other. For one-to-many relationships we need to keep the data in separate tables, and refer to the “one” item in our “many” table. Finally, for many-to-many relationships we do not have a good way to link directly between tables without violating normalization methods; instead we will create small connecting tables where each record represents a valid combination of the items from our two tables.

Primary, Foreign Keys

To find information in our database we need to be able to uniquely identify the record(s) we want to interact with. This can be done several ways. First, we can identify a piece of information from our table that makes each record unique, like a social security number identifies a US citizen. Sometimes, however, our record does not have one single piece of information that does this. Take

a home address for example. To make an address unique, we need to take the street name, number, city, and zip code at a minimum. We can use this method to create a key that uses more than one column to identify a record, called a hybrid key.

Our last method is to let the database make a key for us, so every time we insert a record it receives a number unique for that table automatically. In this method, we let the database manage an auto-increment for that column. While it identifies a particular row, it does not contribute information that relates to the data it identifies. We will see examples of primary keys come into play as we normalize an example dataset, so for now you just need to keep the concept in mind.

4.2: Data Relationships is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

- **4.2: Data Relationships** by [Michael Mendez](#) has no license indicated.