As we work our way through relational theory and go over relational algebra examples, you should be aware of the fact that name qualification is essentially unlimited. The guiding principle to follow in all cases is simply to make sure that in any table (i.e., relation) that you create, including those that result from applying relational operations, the names of the attributes within the resulting table are all unique. The basic approach we’ll use is to place the relation name followed by a period in front of an attribute name drawn from that relation whenever there could be possible confusion. The origins for this approach are unclear, but this approach may very well have been motivated by a similar approach utilized in SQL. As an aside, we have to use qualified names when required to avoid confusion, but it’s our option to use them or not use them when circumstances do not require their use (most people would opt not to use qualified names unless they are required).

Consider a series of examples. Suppose we start with two relations, A and B, that both have three attributes named x, y, and z. This situation would certainly be legal because we only require that attribute names be unique within a single relation, not within an entire database consisting of multiple relations.

If we were to take the product of A and B and name the resulting relation C, then C would have attributes with names A.x, A.y, A.z, B.x, B.y, and B.z (assume for the moment that we do not rename the attributes during the process of assigning the result to C). We had to do this in order to create a relation C that contained uniquely named attributes. Also, our convention is that when we utilize attribute names from the relations involved in some operation, we systematically take the attribute names in an order that corresponds to the order in which the underlying relations are involved. In other words, had we said that we took the product of B and A (with B listed first) to create some resulting relation Z, then the first three attributes in the resulting relation Z would have been B.x, B.y, and B.z.

Now suppose that there is another relation D with attributes named A.x and B.z (most likely, we derived this relation by applying one or more relational operations to some other relation).

If we take the product of C and D and name the resulting relation E, what are the names of the attributes in E? Since A.x and B.z are attributes that occur in both C and D, we need to use an additional level of qualification to distinguish the attributes. So, we wind up with a relation with the following attributes:

C.A.x  A.y  A.z  B.x  B.y  C.B.z  D.A.x  D.B.z

We could continue in this fashion, but obviously more and more levels of name qualification make referencing individual attributes more and more cumbersome. Hence, this is part of the motivation for providing a facility that allows us to rename attributes. (As an aside, the examples shown would seem to adopt an approach you might term full name qualification since whenever there was a potential conflict, we qualified all names involved. As a further aside, we recognize here some issues that we sometimes find when using traditional programming languages. The proposed solution in those cases is comparable.) Once you start to cover actual database languages, such as SQL, you’ll also see that these languages typically provide
mechanisms for dealing with this issue. Most of the time we can find reasonable ways to address this matter, without having to “work” very hard to do so.