

# A Note on the Data Modeling Process

Sudarshan S. Chawathe

Computer Science Department, University of Maryland, College Park, MD 20742, USA.

chaw@cs.umd.edu

October 15, 2003

This note provides an illustrative example of the exploration and decision-making process in the early phases of data modeling using the Entity-Relationship (ER) model. For concreteness, we will use the version of the ER model described in [1].

Suppose we wish to build a database-backed Web application that makes it easy for university students to find other students with shared interests. For example, a sophomore may use our application to find other sophomores who are interested in skiing and live on campus, perhaps to plan a trip.

We begin by noting that we would like to permit searches based on several obvious criteria (age, sex, year, major, etc.). Since all these properties relate to a student, the entity *Student* depicted in Figure 1 readily suggests itself. It is not difficult to determine a primary key for this entity set. In Figure 1, we have chosen to use an artificial key *UID* (a system-generated identifier not to be confused with the university's student identifiers or social security numbers).

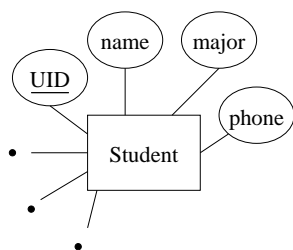


Figure 1: A start to modeling

Since our application is centered on personal information, it is tempting to conclude that we have completed the modeling task for practical purposes: Each person in our application is mapped to an entity in the *Student* entity set. Such entities are uniquely identified by their *UID* attributes and all other attributes relevant to our application can be simply attached to the *Student* entity set. However, closer examination reveals several flaws in the model (or, equivalently, restrictions on the application that are stronger than desirable).

Let us look more closely at the phone attribute. As modeled by Figure 1, the database stores exactly one

phone number for each student. (If we permit nulls in the resulting relational schema, we may accommodate students with no known phone number; however, nulls lead to other problems.) If we are willing to live with this restriction, it is certainly not incorrect to model our data in this manner. However, should we wish to design a more flexible database that permits several phone numbers to be stored for a student, some modifications are needed. Perhaps the simplest step to take next is to promote phone numbers from attributes to entities belonging to a *Phone* entity set, as depicted in Figure 2. A relationship *PhoneUse* is used to associate student entities with the corresponding phone entities.

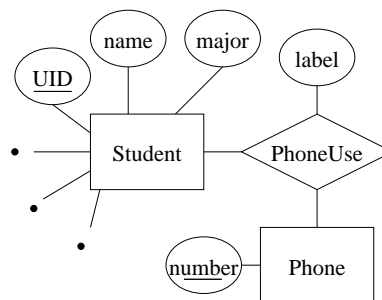


Figure 2: Allowing multiple phone numbers

The *PhoneUse* relationship suggests further investigation of the nature of the mapping between students and phones. In the current form, the relationship is simply a many-to-many relation between the *Student* and *Phone* entity sets. This design, which permits, for example, a phone to be shared by several students (perhaps a shared office number) is suitable for our domain and therefore we do not introduce any constraints on the multiplicity of *PhoneUse*. However, we can do better than to provide our users with simply a set of phone numbers for students they wish to contact. An indication of the type of each phone (e.g., labels such as home, office, mobile, emergency) would improve the utility of our application. This feature is easily accommodated by adding a *label* attribute to the *PhoneUse* relationship. Attaching this attribute to the *PhoneUse* relationship allows us to tag each student-phone pair with a suitable label

instead of using only one label per phone (as would be the case if the label attribute were attached to the Phone entity set). Our design permits, for example, Alice’s home phone number to be Bob’s emergency number.

At this point, we may be reasonably satisfied with the modeling of phone numbers and their association with students in our domain. However, there are further enhancements that certainly make sense. Although the information provided by the label attribute of the PhoneUse relationship provides some guidance on the phone number to use to reach a student, it is far from ideal. For example, if the database lists both a home phone and a mobile phone for Cathy, which one is a better choice for calling her at 7:00 p.m.? Perhaps Cathy would rather not divulge too much information in this regard, but we may decide to permit such information to be stored in our database for users who wish to provide it. A simple solution is to add another attribute to the PhoneUse relationship for the purpose of describing times at which a student prefers to be reached at a certain phone number. A single attribute is not a good choice for this purpose because it would be difficult to represent even simple time intervals (e.g., 7:00–9:00 p.m.) in a single attribute. (While encoding intervals into strings formatted in a predefined manner is feasible, such a design is not in first normal form and leads to problems in queries.) A single interval may of course be easily represented using a pair of attributes. However, we may again wish to design a more flexible database that permits arbitrary sets of intervals to be associated with each student-phone pair. For example, Bob may want his office phone to be used during 10:00–11:00 a.m., 12:30–3:00 p.m., and 4:00–6:00 p.m. This feature requires that we associate each student-phone pair with a set of intervals and suggests the creation of a *TimeInterval* entity set to model time intervals. This design is suggested by Figure 3.

We have chosen to model TimeInterval entities using a pair of attributes representing the start and end times of the intervals they represent. Note that both attributes are key attributes. This design is probably sufficient for our application but it is easy to note that it too has several straightforward extensions. For example, we have implicitly assumed that all times are expressed using some common standard (perhaps Eastern Standard Time). This assumption is quite reasonable if our application focuses on a single geographical area. However, if our application were to span multiple timezones, and especially if we wished to use the TimeInterval entity set for schedul-

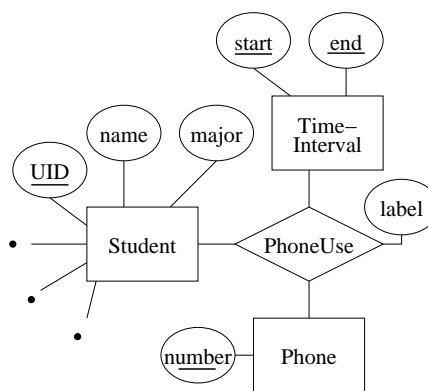


Figure 3: Adding time-of-use information

ing meetings among students, it would be prudent to model time more carefully. Such modeling would almost certainly be overkill for our original application. Part of data modeling is determining which concepts need to be modeled and at what level of detail.

Most of the above discussion focused on what began as a single attribute of the Student entity set. It is not unlikely that other attributes and entity sets will lead to similar issues under closer scrutiny. For example, we have modeled phone numbers using a single number attribute. However, splitting a phone number into its components yields some benefits and, unsurprisingly, drawbacks. As a simple exercise, the reader may wish to extend the design to include information about meetings among students, including both scheduled (future) meetings and past meetings.

Hopefully, this note has illustrated that even a seemingly simple application requires careful thought to arrive at a data model that is general enough to permit the application to function in the desired manner without being overly general and cumbersome. It is important to understand the restrictions a model places on the kinds of information that can be modeled. Restrictions that are not reasonable for the application must be removed using a process similar to one described above.

**Acknowledgment** This note was prompted by a discussion with Richa Singh and Chris Schneck about their class project this semester. The errors are mine, of course.

## References

[1] J. D. Ullman and J. Widom. *A first course in database systems*. Prentice-Hall, Upper Saddle River, New Jersey, second edition, 2001.