1 Introduction to Modeling Methods

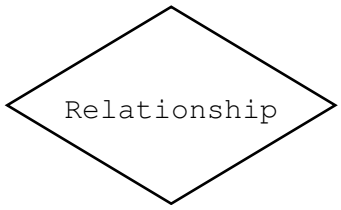
When designing and specifying databases, it is convenient to have a *visual representation* of a database schema. In this document, we will discuss two primary methods: Entity Relationship Modeling (E-R), and Unified Modeling Language (UML).

2 Entity Relationship Modeling

Entity Relationship (or E-R) Models are relatively old way of modeling databases. Just about nobody uses them anymore[1]. In any case, it is still very useful to know and understand the modeling concept.

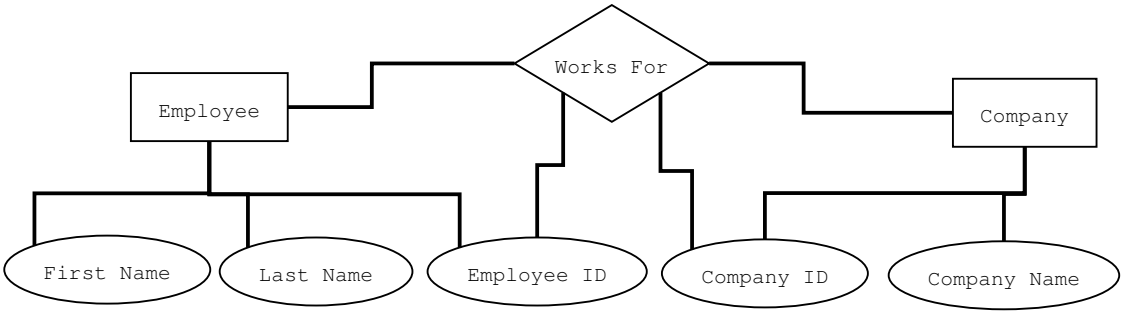
E-R models consist of entities, relationships, and attributes. We represent entities via boxes:

Relationships are represented via:



There is another artifact, which is an 'attribute'. Entities and relationships may have attributes. It is represented by an oval:

A model emerges if we put all of those together into a single picture:



Notice that "Employee" has first name, last name, and employee id, while "Company" has company name, and company id. The "Works For" relationship has employee id, and company id.

Basically the idea is to make a picture that unambiguously describes a database schema. You should practice drawing E-R diagrams for all the databases you create in this class.

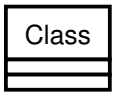
[1] Almost everyone has switched over to using UML.

3 Unified Modeling Language

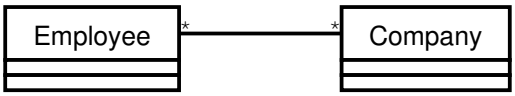
UML, or Unified Modeling Language is probably the most popular modern way to modeling databases. It is a combination of many modeling languages, and is extensible. In fact, if you want to add something to the picture, just go ahead and add it in.

The language was specified by OMG (`http://www.omg.org/`), and has been implemented by numerous tools, such as Rational Rose, etc. It is important to note that the standard and the tools should only be used as a guide, not the absolute truth. UML does not create databases for you—it is still your job. It is only a pictorial language you can use to help you figure out what goes where.

There are many flavors of UML meant for different tasks. For our purposes, we'll be mostly concerned with "classes" various relationships they can have. A class is represented by a box:

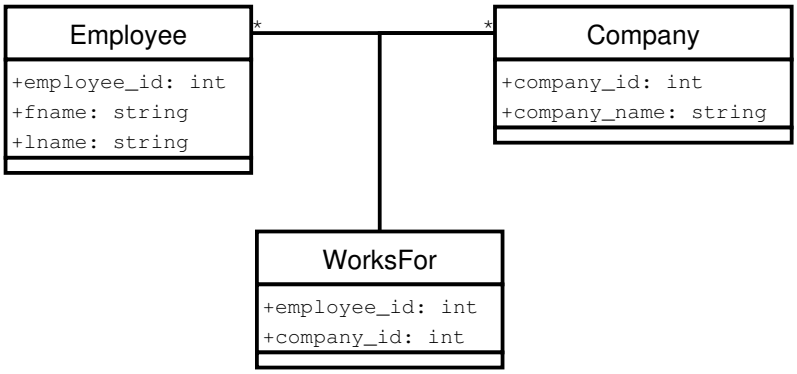


The relationships are just lines or arrows between these classes. For example, our E-R example can be re-drawn in UML via:

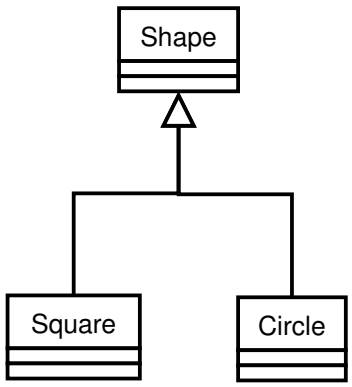


Now, obviously this doesn't give us enough details to actually create a database.

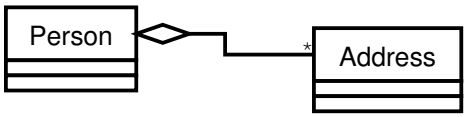
One of the best features of UML is that we can specify as much or as little as we want. This is very useful for the design stage. For example, we know we will need to have an Employee object/table, and we know we will need to have a Company object/table, but we don't know their respective fields yet, or how they'll be linked together. Once we learn more details, we may mutate that diagram into something like this:



You can also represent inheritance via the 'generalization' (or 'specialization' in reverse) arrow:



There is also aggregation, which is an inclusive association—where one class is part of another class. For example, a "Person" may have an "Address". Now, it doesn't really makes sense to have an address all by itself (well, assume that it doesn't), so the "Address" belongs to a "Person". We can represent that via:



3.1 Problems with UML

The major problem with UML is human related. We tend to over-design things, and UML is a perfect language for over-designing things.

Another quasi-problem is that UML is extensible. Everyone supports the default set artifacts (like class, etc.), but some of the fancier ones aren't very standard—there are things in Rational Rose that are not in other UML packages. So try avoid less-than-standard things in your models.

Yet another problem is that UML isn't precisely a relational modeling language. It works great for databases, but what it does best is model "object" oriented system. For example, it models *inheritance*, even-though it's a major pain to do in a relational database (we'll discuss some methods a bit later).

3.2 Tips for using UML

UML is a great language for designing applications, and pretty much anything. The best tip I can offer is to use a dry-erase marker on a white-board, or chalk on black-board, or pen & paper, etc., to draw UML models, and *not* some UML tool. Those are great for documentation, when you already know what you want. They're usually pretty bad for the brain-storming part of the project.

Understand everything that you're doing, and pay particular attention to how things will work out in a relational database. For example, do your best to avoid inheritance like structures, etc.

3.3 Generative Programming

UML happens to be one of the very few languages that can greatly aid generative programming. The idea is that most database applications are the same (or similar). Every time you develop a database application, you're doing the same thing over and over again, repeating what you did on your last database project, etc.

If you structure your project around code generation from a UML schema (or UML like schema), you can save yourself a *lot* of time.

To get a better idea of what I'm talking about, consider this document:
`http://www.theparticle.com/documents/genprog.pdf`