Jessica Rieger

1. One database in the world today is the record of all the medications in the world and who is taking them. On a smaller scale we could just say 1 pharmacy's record of medications and who takes them. The elements of data within this database are the numbers regarding how much of each medication to take, the name of the medication, the people's names, the dates associated with when the prescription was last filled, and the number of times they have left to fill the prescription. Even in presenting this list, it is hard to describe the data without hinting at some context because without it I would just have to say numbers, words and names and that would be pointless since you would not know what I was talking about. Clearly, we have information in our database and not data. The data of numbers and names becomes information when we organize it based on the person's record or combine all the data surrounding a particular medication. By grouping the data in these ways we give it context and avoid ambiguity, which is by definition how to make information out of data. For example, if we simply say 75 it is completely ambiguous. This is a piece of data, but with no context surrounding it, it is useless to you. When it becomes information by saying perhaps 75mg of Tylenol per hour for infants, we then have context and the data (now information) becomes useful. We can even go a step further and say who exactly should receive the medication and for how long, etc. By adding the details to surround the data it becomes more and more useful to us. In this case, once we have information, we can accurately treat patients, get them the correct medication in the correct doses, and avoid confusion over the meaning of numbers. On a more general scale, once data becomes information we can use it to do analysis, simply draw from it as an accurate reference, or store things for later that will have meaning and not simply be pointless.

2. The Hierarchical Data Model, or the Information Management System, is one in which all parts of the model descend from a single root node. There are branch children that come off of the root node which in turn have their own children below them. This model allows us to see the structure of whatever we are building in a very clear way. This model does solve a few of the problems of its predecessors, namely that it establishes itself with physical data independence by being separate from the DASD, yet it still has several problems. First, in order to write the code for a structure such as this, we must know what it looks like in advance instead of being able to simply inquire what it looks like as we go along. Furthermore, it does not allow us to share different leafs between branches. In our game example, this resulted in 2 instances of item B, which would clearly be problematic for our game since they could essentially be different things labeled as the same name and the program wouldn't know that was wrong. Finally, there is nowhere to put things that aren't yet associated with a branch. In our game example this becomes problematic because we have nowhere to represent items such as D that are owned by no one.

The other model is the Network Model. This model is similar to the Hierarchical Data Model, but it eliminates the redundancy of B previously mentioned by allowing branches to connect to the same leaves. Other than that though, this model is very similar to the Hierarchical Model and has several of the same problems. As before, the problem of needing to know the structure in advance is ever prominent as well as the lack of a place for things that don't belong under any of the branches. We can circumvent the latter problem by creating a branch for unknown things however in the long run this would not be very effective. Furthermore, the Network Model and Hierarchical Model are difficult to interact with since any interaction desired requires the use of code writing. Lastly, these models are not traffic cops, meaning that they have no idea if there is more than one thing under the same name. In relation to our game, this means that 2 payers could have identical names and the program would not care. Clearly, though these models were advancements on the prior Flatt File Model, they have a long way to go until they are suitable for our needs.

In terms of the XML Database Model, I think that it has its own pros and cons as most models do. From what I have read regarding this model, it seems as though it can be of equal caliber to the Relational Database Model depending on the information you are trying to store. It does have some positive features that seem as though they would be really useful. Particularly, the fact that almost any data can be expressed as XML gives it a versatility that relational models lack. Furthermore, the XML database is said to be "self-describing," thus it can be read by anyone and sent to others as an entire package. The fact that the data does not need to be read by an expert makes it more appealing as a way of storing data in the business world where they want answers quick. Finally, on a more technical note, the database can be searched without knowing the static definition of the schema meaning it is easier to find what you are looking for or make changes when storing relatively small amounts of data.

However, as is usually the case, there are some shortcomings of this model. In terms of efficiency, it is much slower than the relational model due to the parsing and interpreting required. The emphasis on speed within our culture makes this shortcoming prominent. As well, if you do not need to know the child/parent relationship to use the data/information, then this model seems as though it would require unnecessary work. Finally, if you are storing large amounts of data, this model is inefficient and unreliable since there is no mature management system to retrieve data.

Overall, I think that this model sounds better as a tool to help with relational databases (by making the XML databases out of relational queries.)   In the world today, so many of our databases contain huge amounts of data so I think that this model would struggle to keep up. Furthermore, its structure, though helpful, makes it difficult to see the relations among categories that are far apart compared to the relational model where the arrows clearly show these relations. In truth though, it all comes down to your database needs and which model will fit them best.

Sources: http://xml.coverpages.org/IBM-XML-GC34-2497.pdf