

Modeling Concepts vs Modeling Data.

Thanks to Dr. Ted Codd, the constructs of a relational model have a strict mathematical rigor to them, one based firmly on set theory and the theory of relations and functions. When given an interpretation, these mathematical constructs become the basic ontological categories of the subject matter being studied. When given an implementation, they become the tables, columns, foreign keys, functional and multi-valued dependencies, and the entity, referential and domain constraints for a specific database.

So what more is there to data modeling than this? If one can normalize to 5NF and still be only a journeyman modeler, what more is required to be an expert modeler? As I said before, the difference is this: the expert modeler models concepts, not just data.

But “concept” is itself a difficult concept. Computer science curricula ignore it. This is evidenced in the computer science literature, which is almost without exception concerned with data and not with concepts – how to store and retrieve data, how to move data about, how to manipulate data to generate new data, how to present data and, in general, how to do all of these things for maximum value and at minimum cost. Computer Science programs, reflecting their origins in university Electrical Engineering departments, train *engineers* – not *semanticists*, and not *ontologists*.¹

¹ The awareness among computer scientists, however, of the value of ontology and semantics, is beginning to improve. See, for example, the work of the IEEE 1600.1 working group on a Suggested Upper Merged Ontology.

For example, a computer science article might use an Inventory table, and a stream of transactions updating that table to illustrate count filters, and an improved variation on them.² But clearly, if the table and transactions used for illustration were about a Sales Receipts table and the transactions that update it, that wouldn't matter. The difference in subject matter is no difference at all. The tables and transactions are given a real-world interpretation only for mnemonic purposes, since many of us have difficulty following a purely formal engineering discussion, replete with the mathematics that renders the discussion precise, and that points in the direction of actual solutions that could be implemented in data schemas and code.

Business IT departments also focus almost exclusively on data. But occasionally, they will find themselves employing or hiring on contract a data modeler who asks questions like “What do you think a customer *is*?” or “What do you *mean* when you say that these two products, produced at two different plants, using different bills of material, are the *same* product?”.

On the one hand, there is a tendency to believe that a person who asks questions like these must be really smart, for these are clearly “deep” questions. On the other hand, the project has a deadline, the business users have pointed to the source tables that the new database will replace, and the project manager wants a fully normalized model of the same data that is in

² J. Aguilar-Saborit, P. Trancoso and V. Munes-Mulero. “Dynamic Count Filters”. *Sigmod Record*, vol.36, #1 (March, 2006), pp. 26 – 32.

those source tables, and he wants it in a hurry. “Deep” questions are fine, but when the schedule gets tight, the project manager may find himself wishing for a simple, straightforward journeyman modeler. Let someone else, on someone else’s budget, worry about what a customer is!

So what is the value of these “deep” questions? Why isn’t a fully normalized and data-element-complete model good enough? What is “advanced” about an advanced data model?

I’m not going to answer these questions here. For now, it’s enough to have posed them. Remember, good questions are worth their weight in gold; and good *initial* questions are worth even more than that.

Written sometime in 2006.