

Data Modeling As Engineering.

Dr. Tom Johnston.
MindfulData.com

Computer science literature is dominated by writings on data engineering. By that I mean that the topics discussed are data structures and how to manipulate them. What those data structures are about is incidental. Mathematics abstracts from such considerations, and what data engineers are after is new mathematics to better manipulate data structures.

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.¹ 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.²

This is also true of all discussions of relational data normalization that I know of. Chris Date's is the classical and probably best-known one, and his example is a database of Suppliers, Parts and Customers. But clearly, he could have made all of his points with a database of Students, Teachers and Courses (another favorite of those who write about databases but do not actually design real-world ones).

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

² Although a principal theme of this book is that real-world data practitioners need to understand the concepts behind the data they model, and behind the code that transforms values of variables in accordance with precise rules, and not just the mathematics of manipulating abstract structures, my comment here is not sarcastic. Pointing in the direction of implementation of useful data schemas and transformations is the bottom-line practical value of discussions couched in the language of data engineering. (The bottom-line un-practical value is the beauty of the patterns described/discovered – which is not to say that beauty is not an excellent indicator of what will ultimately be of practical value.)

A brief anecdote will illustrate my point. A friend of mine telephoned a computer science department that was one of the top ten in the country. It turned out that the department chairman – a man of the stature that Turing award winners are selected from – was walking by as the phone rang and answered the phone himself. My friend explained that he wanted to enroll as a graduate student, and the first thing the chairman asked him was “What was your math score on the GRE exam?” My friend had made quite a high score, landing him well within the top ten percent, and he told the chairman what that score was. The chairman responded, “I don’t think you should apply here. We generally look for much higher math scores.”

Insofar as computer science is an engineering discipline, this answer is perfectly understandable, and even appropriate. But if computer science remains exclusively an engineering discipline, I don’t know who will train data practitioners in analyzing the *meaning* of the data they manage. Which brings us to a different way to look at data modeling.

Data Modeling As Conceptual Clarification.

Before the mathematics is what the mathematics is about. If we are normalizing a set of tables with a salesperson-id and a region-id as primary key, and ytd-sales-amt as a non-key column, the data modeler is encouraged to ask the question “Is ytd-sales-amt functionally dependent on both salesperson-id and region-id?” If the answer is “yes”, the table is normalized; otherwise it’s not.

(insert illustration model, but no sample data case)

But functional dependency is not the basic issue. It only formalizes a relationship which, if it exists, exists because of what the columns *mean*. In particular, the fundamental question, rather than being about functional dependency, is actually about the meaning of the concept ytd-sales-amt. In the context of this example, that concept could mean three things, namely:

1. The ytd amount sold in that region.
2. The ytd amount sold by that salesperson.

3. The ytd amount sold, by that salesperson, in that region.

The functional dependency in question exists if and only if “ytd-sales-amt” *means* the third thing. A skeptical reader – which I encourage you to be – might suggest that the difference here is little more than a difference in terminology. Good data modelers, she would say, end up with the same result whether they think in terms of functional dependency or in terms of meaning.

I intend Part I to support my contention that asking “What does ‘ytd-sales-amt’ mean?” is not just a terminological variation on the language of functional and multi-valued dependencies and the other terms in which the mathematics of relational theory and relational normalization is expressed.