

Data Modeling and Ontology: the Past, the Present and the Future.

Dr. Tom Johnston.
MindfulData.com

Mathematics, Ontologies, Concepts and Deep Questions.

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, as representations of real world concepts and their instances, 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

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

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.

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

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 "In this company, what *is* a customer?" 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 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.

The Present.

As far as the management of data is concerned, relational data modeling is the present. Data modelers define an ontology for a subject area when we define a set of tables for it. In doing so, we attempt to express the semantics of what it is about that subject area that our company is interested in.

The Mathematics of Relational Theory.

Like all inferencing engines, relational databases require (a) a syntax into which we must translate our semantics; (b) a set of rules distinguishing grammatical from ungrammatical syntactical structures; and (c) a set of rules for transforming syntactical strings into other syntactical strings. This is all provided by the mathematics of relational theory, especially the mathematics of set theory, functions, relations, propositional logic and predicate logic.

Relational Normalization.

The theory of normalization is also an essential part of relational theory. It is usually presented as an exercise in applied mathematics. By understanding functional dependencies (and eliminating partial and transitive ones) and multi-valued dependencies, we can create a fully normalized set of tables.

What is *not* well understood is that normalization is an exercise in *applied semantics*, albeit an applied semantics with a mathematical base. It is an exercise in recognizing and eliminating homonyms and synonyms, a process which in turn presupposes that the semantic anomalies of ambiguity and vagueness have already been eliminated from the universe of concepts being modeled.

The main purpose of normalization is to eliminate redundancy in databases. An unnormalized database, by definition, contains redundancy. But not all fully normalized databases are free of redundancy. Since redundancy is the source of inconsistency within a database, it is important for us to understand the ways in which redundancy may remain even in fully normalized databases. (See my four articles "Unobvious Redundancies". Links are provided under the Publications tab.)

Early Binding in Relational Theory.

Although computer scientists are moving beyond relational theory, to create even more powerful inferencing engines, there is one serious flaw in relational theory which I think is not well understood, and cannot be left unaddressed. It is not some incidental feature. Rather, it is the early binding of semantics to syntax, and it is based on two features of relational theory. They are:

1. The use of business data in primary keys.
2. The use of foreign keys to express relationships.

Figure 1: Two Early-Binding Features of Relational Theory.

The use of business data in primary keys follows directly from Dr. Codd's Information Principle. As re-affirmed recently by Chris Date, this principle lies at the heart of relational theory.³ The use of foreign keys to express relationships also follows from the Information Principle, since without them we would need some kind of DBMS machinery, some kind of internal pointer mechanism, to express relationships.

With these two features, relational theory “early binds” much of a database's semantics to its syntax. This is ironic, since one of the ways in which relational theory claims to be superior to pre-relational ways of managing data is in providing a high degree of *data independence*. Data independence is precisely the separation of semantics from syntax, the “late binding” of the former to the latter (although it is not often expressed this way).

It is a great irony that *the ways in which relational databases early bind semantics to syntax are the single greatest source of cost in the maintenance*

³ Get references to both Codd and to Date.

and enhancement of IT systems. (See "Primary Key Re-Engineering Projects", under the Publications tab.)

Ontology and Relational Theory.

It is fairly well known that there can be more than one fully normalized model of the same subject area. What is not well understood is how to choose the best one of them. I will present my account of how to do that.

The entities of a relational data model represent the *ontological commitments* made by that model. They are the kinds of things, the categories of existence, which that model says there are. We will examine this point in terms of W. V. Quine's criterion of ontological commitment, which is – rather famously – this: to be is to be the value of a variable. We will see that SQL DML (Data Manipulation Language) statements are statements of first-order predicate logic whose variables range over the tables of a database.

Tables are usually defined in SQL DDL (Data Definition Language) statements, especially the CREATE TABLE statement and its various clauses. But this way of defining tables early binds their semantics to their syntax. If that semantics ever changes, the cost of unbinding it from the old structures and re-binding it to the new ones, can be incredibly expensive – far and away the single most expensive kind of cost that IT departments incur.

But semantics can be expressed in code as well as in data structures – in the rules that manipulate well-formed instances of syntactical structures as well as in those structures themselves. For example, in formal systems of logic, the rule of modus ponens (If P implies Q, and P is true, then Q is true) can be expressed either as an axiom or as a rule of inference. Expressing it as an axiom is equivalent to using a CREATE TABLE statement. Expressing it as a rule of inference is equivalent to writing it in code.

However, code is also early-bound; it is, specifically, compile-time bound. (For SQL statements, this means bound at the time the statement is written.) The advantage of expressing semantics in code over expressing it in data structures is not binding time. It is simply this asymmetry: you can change code without changing data structures, or other code; but if you change data

structures, you must change all code that references them. So, bottom-line, it's much easier to change code than it is to change data structures.

A third way to express semantics in an IT system truly late binds the semantics, to either data structures or to code. It is by using tables of *active metadata*. By that term, I mean metadata that gives a specific interpretation to tables, or that alters what already-compiled code does.

The final way to express semantics in an IT system is to leave it up to the user. But as long as what an IT system presents to its users does not “wear its meaning on its sleeve”, i.e. as long as what it presents is not free of ambiguity and vagueness, and is understandable by all its users, then it is a mistake to “leave it up to the user”.

These related issues of binding time, and the form in which the semantics of an IT system are expressed, are the ways in which we can choose the best among a set of data models all of which are fully normalized, and the best among a set of information systems all of which meet the users' requirements.

The Past.

As for the past, ontology has its roots in the philosophy of Plato and Aristotle. It begins, for our purposes, with Plato's theory of Forms, Aristotle's response in his theory of matter and form, and especially Aristotle's distinction between essence and accident. Although no longer widely accepted, I have found the distinction between essence and accident very helpful in determining the proper keys for relational tables.

Aristotle's works *Categories*, *On Interpretation* and *Prior Analytics* represent mankind's first attempts to create a top-level ontology. After reviewing this work of Aristotle's, which we will later compare to upper level ontologies that are being developed by the computer science community, we will turn to Aristotle's logic, especially his theory of syllogisms. This is the first attempt – and a highly successful one – to create inferencing engines based on logic. (Inferencing engines based on

mathematics had, by that time, already been developed, by the Indians, Babylonians, Egyptians and by earlier Greek thinkers.)⁴

In addition to this focus on Aristotle, we will briefly review the history of universals, especially the doctrines of realism and nominalism as formulated in the Middle Ages by such philosophers as Aquinas, Duns Scotus and Ockham. This has only peripheral relevance to relational data modeling, but I am including it because it is very relevant to object-oriented theory and the distinction between classes and objects. The peripheral relevance to relational data modeling, of course, is the parallel between classes and both domains and entities, on the one hand, and between objects and both domain values and entity instances on the other.⁵

As for semantics, it is best reviewed from a current concepts point of view, not an historical point of view. I will take as my guide here John Lyons, whose many books on linguistics are classics in the field.⁶

The Future.

The future of ontology and semantics, as far as the management of data is concerned, is in good hands. The computer scientists who are doing this work are following in the footsteps of Dr. Ted Codd, who was the first to gather together the threads of mathematics, set theory and formal logic, and use them to weave the fabric of a powerful inferencing engine – the relational theory that is the foundation of today’s relational Database Management Systems (DBMSs) and relational databases.

Over two millennia of work in ontology and logic is relevant to this future. What’s different about ontology, as done by today’s computer scientists, is that the focus is on inferencing engines, on formal languages that will support automatic inferencing (in the form of string manipulation) with ever-increasing semantic reach. Aristotle was the first to develop an integrated set

⁴ A good source is D. E. Smith, *History of Mathematics* (Dover Books, 1958; first published 1923).

⁵ Although Chris Date has strenuously argued that classes should be compared only to relational domains, and not to entities, and that objects should only be compared to domain values, and not to entity instances.

⁶ *Linguistic Semantics: an Introduction* (Cambridge, 1995), *Introduction to Theoretical Linguistics* (Cambridge, 1968).

of ontological categories, and the first to develop an inferencing engine based on logic. But for him, these were two separate disciplines. For today's computer scientists, these two disciplines are tightly integrated.

This work is slowly making its way into business IT. First-generation commercial software is now available that will help IT departments create and manage controlled vocabularies, taxonomies and even ontologies. Work by various industry consortia is leading to standardized vocabularies, taxonomies and ontologies in those industries and industry segments. To the extent that databases are managed on the basis of these standardized components, those databases will be "semantically interoperable".

This means simply that queries which range across those databases will return meaningful results. If I run a query against my own company's databases looking for medical claims that meet certain search criteria, I am pretty sure of what my result set means. That's because I know (or should know) what counts as a medical claim, i.e. what criteria are met for a medical claim to be represented by a row in my Claims table. And because I know what the search criteria mean.

But in today's world, I would *not* be confident in running that query across both my own company's databases, and (probably view-mediated) claims-related databases in client companies, and other claims suppliers. It is highly unlikely that those other companies mean exactly the same thing by "medical claim" that my company does. For example, my company may keep a row in the Claims Header table even for claims that lack any claim lines. Other companies might filter out those claims. If we both started with empty databases, and loaded them from the same data sources, my count of claims would differ from their count of claims. This is what I mean by saying that we don't *mean the same thing* by a "medical claim".

Semantic interoperability means that we do mean the same thing. Without it, a query across databases of multiple companies is likely to produce a result set of "apples and oranges". By building our databases on the basis of a common controlled vocabulary and syntax, and by employing a standard ontology as at least the upper level set of tables in our databases, we insure that queries across our databases that reference those common tables are referencing semantically identical concepts. Such queries will return consistent results, not apples and oranges.

Therefore, we will concentrate on three topics, and on how they are making their appearance in business IT departments. Those topics are:

1. Controlled vocabularies;
2. Taxonomies; and
3. Standard ontologies.

Figure 2: Three Ways Semantics and Ontology Are Making Their Appearance in Business IT.

But we should also see a little beyond the near horizon of our own practical interests. Besides knowing enough about these topics to better design our own databases, we should also know a little about what is going on at the leading edge of this work. To illustrate this, I will discuss John Sowa's semantic networks, and his notions of "knowledge soup" and a "lattice of ontologies".⁷

The past, the present, and the future. Relational data modelers, working in business IT departments, are a part of a long and honorable tradition. We are doing applied ontology and applied semantics, and so we should know something of our intellectual history. More than just being appropriate, this knowledge has practical value. It will help us do our jobs better. That is what I hope to demonstrate with this book.

⁷ *Principles of Semantic Networks* (Morgan-Kaufmann, 1991), *Knowledge Representation* (Brooks/Cole, 2000).