

## Ontology: Things and Events.

Dr. Tom Johnston

MindfulData.com

Written sometime in 2006.

{  
*This is the amalgamation of three sets of notes. As a result, there may be some inconsistencies in terminology. I'll revise this material later on, and clear up any such issues. But for now, I think this essay still has value, even if there are some terminology problems.*

*What is that value? It is the value theory always has. It is a framework for thinking about things which organizes our experience by revealing patterns among things and events (or, as other philosophers would have it, by imposing patterns on things and events).*

*The patterns that ontology is concerned with are deep patterns. This term, "deep patterns" is both a description and a compliment. As a description, deep patterns are distinguished from what we might call "surface patterns" in several ways.*

- 1. Surface patterns are pretty obvious; deep patterns are not.*
- 2. Deep patterns constitute the theory of something of interest; surface patterns comprise the engineering through which the theory is applied.*
- 3. Familiarity with surface patterns supports treatment of a difficulty. But only familiarity with deep patterns will support diagnosis.*

*This last distinction is familiar to we IT professionals. Consider a software application that has been around for a long time. Multiple releases; countless bug fixes between releases. Often, an IT shop will have one person who is the guru for that system. And what distinguishes the guru from the other development and support personnel is not how well or how fast he can write code, or even how elegant his code may be. It's that the guy (consider "guy" a gender-neutral term) has an uncanny ability to figure out what's wrong when the system crashes, or when it produces crazy results.*

*I think most of us with a solid background in systems development and maintenance would agree with this description. There may be lots of very smart programmers who have worked on the system. But there's something almost mystical about the guru, and almost magical about how often his diagnostic hunches are exactly right.*

*So what does the guru know that the other smart guys don't? He knows the system. The other guys know a bunch of programs. Even if they are familiar with all the programs, and with what each one does, they don't have that gut-level instinct that the guru does.*

*Instinct? It's got to be something more than that because it reveals a knowledge of a logical structure (however convoluted) – a collection of algorithms being applied to*

*instances of a data structure. The guru's ability may look like instinct; it may even feel to him like instinct. But it's clearly got to be something more than instinct. This something more is a grasp of the deep structure of the software application. It is a grasp of the underlying theory of the system.*

*Why, then, can't the guru clearly explain, step by step, how he thought through the problem and arrived at the correct diagnosis? Because to do that is a different skill. It's called pedagogy. It's the skill that makes a good teacher, or a good author.*

*The theory is a pattern. Patterns aren't grasped piece by piece. You either see a pattern, as a single, complex thing, or you don't see it. The guru sees the pattern "in his mind's eye". In very rare circumstances, the guru can also explain what he sees, but an inability to explain it does not imply a diminished ability to see it clearly.*

*So: this is why theory is important. What does this have to do with ontology and IT? Just this: in my view, there are a sufficient number of speakers and authors around who will explain the surface patterns of ontology as it applies to the management of data. They will explain the engineering of it – RDF triples, OWL, and the intricacies of ontology management tools.*

*This engineering knowledge is all that you need to be the kind of smart programmer I talked about. But it will not make you a guru. For that, you need the deep structure. You need to recognize the patterns for which the engineering knowledge provides a toolkit of tools that manipulate those patterns. In the hands of a journeyman, if overseen by a master craftsman, these tools will produce beneficial results. But that is because, aware of the deep structure, the master craftsman decides what needs to be done – how to build the most beautiful construct that will solve whatever practical problem is at hand, or how to repair an existing construct buffeted and broken by unexpected storms, or how to integrally create a completely new addition to the building.*

*The way to become a master craftsman of the application of ontologies to data management, to become a guru, lies first in an understanding of ontology itself, a subject that reaches back almost three millennia, and whose gurus came, originally, from the Aegean islands (especially Miletus) or the Greek mainland (especially Athens).*

*I'm working on providing this historical introduction to ontology elsewhere on this website. Here, I take another approach – to jump in and start doing some ontology. I'll focus on what computer science ontologists would call an "upper ontology". I prefer the opposite metaphor, and call it a "foundational ontology".*  
}

## **Things and Events: a Foundational Ontology.**

First, let's *do* some ontology.

- The world consists of **things, events** and **stuff**.
- **Things** endure over time, and because of that, they can change. They have properties and relationships, and these are what it is about things that change. Following Frege, logicians consolidate this distinction and speak of predicates. Properties are "one-place predicates", a relation between two things is a "two-place predicate", and so on. Things change because they take part in events, or, to put it another way, because events happen to them.
- **Events** are what, of interest to us, happens. Each event happens once, and then it is over. An event can be what happens all at once, at a given instant. Events can also be what happens over a given period of time. In either case, events do not change, even though they exist over time. Since, at any moment or duration of time, even at one place, a lot is happening, "of interest to us" is what picks out what makes up the event. But is it true that events do not change? What about, for example, all the changes going on at an event which is a two-hour rock concert? It is the things in or at that event that are changing, not the event itself.
- **Stuff** is what exists that can't be counted. As a result, nouns for stuff don't have a plural form. Boats are things; there are many of them. But we can't say "Water are things". We also can't say "Waters are things". And this is the way we handle stuff. We handle chunks of it, and chunks of stuff can be counted. Names of chunks of stuff do have a plural form. A lake is a thing. Also, lakes are things. A glass of water is a thing. Also, glasses of water are things. So while stuff isn't a thing, chunks of stuff are things.

## **Things.**

There are things, and there are kinds of things. The kind of thing an object is, is its **type**. Each thing of a certain type is an **token** of that type. A token is also called an **instance**. Thus, the term "customer" denotes a type of thing. Customers Jim, Carol and Fred are tokens of that type.

Types of things are represented in databases as tables. Tokens (instances of those types) are the rows of those tables. This is the start of how ontologies and databases are relevant to one another. A database is an ontology for a "micro-world", for the world of things and events that it represents.

If a database is an ontology, how, then, are events represented in databases? The answer is: just like things. Types of events are represented as tables, and specific events of those types as rows of those tables. I'll say more about how events are represented in databases later on, but for now, a hint: think *transactions*.

Most types have instances.<sup>1</sup> Human being is a type, and you and I are instances. MasterCard cardholder is a type, and I and a few million other people are instances. Invoice – for the Acme Company – is a type, and every invoice Acme has issued is an instance of that type.<sup>2,3</sup>

(to be continued.)

---

<sup>1</sup> The ones that don't are null sets, sets that have no members. Examples include the set of odd numbers divisible by two, and the set of all men born after 1900 who were or are over twenty feet tall.

<sup>2</sup> I have used the fictitious Acme Company as an example in many of my articles, and will do so here also.

<sup>3</sup> Logicians often refer to this as the distinction between types and “tokens”, but I think the word “instance” is a better one to use. I would note also that, in object theory, classes are types and objects are their instances. In set theory, sets are types, and the members of those sets are their instances.