

## ***Bad Data Models and How They Work.***

Dr. Tom Johnston

Originally published: Datawarehouse.com. May 17, 2002.

Permission to re-publish given by the senior editor of DM Review.

If there are good data models then, by contrast, there must be bad data models. And if it's worth taking the time to learn how to build good data models, it can only be because the distinction between good and bad data models makes a real difference, a bottom-line difference.

But data models don't make any kind of bottom-line difference by themselves. Data models define data structures by being translated into DDL and then compiled into the catalog for a database. Only then do they become important. They become important because application code, as well as stand-alone queries, use those data structures to insert, update, delete and retrieve data stored in them.

Let's distinguish between *bad* data models and flat-out *incorrect* ones. An incorrect data model is one that, if implemented, would not allow the business requirements related to that data to be satisfied. For example, a data model that specified a one-to-many relationship between salespersons and customers, when the business requirement was that a salesperson can serve many customers, and a customer can be served by many salespersons, would be an incorrect data model. A data structure with a salesperson foreign key in the Customer table cannot associate a customer with more than one salesperson.<sup>1</sup>

---

<sup>1</sup> Note that the converse – a requirement to represent a one-to-many relationship between customers and salespersons, and a model that supports a many-to-many relationship – is a case of a bad data model, but not an incorrect one. An associative table – the means by which a many-to-many relationship is implemented in a relational database – can record one-to-many relationships. But with this data model, code must compensate. Code must insure that no violations of the one-to-many rule make it into the database.

By the same token, we can distinguish between bad application code and incorrect code. Incorrect code is one that generates invalid updates, or that returns invalid information.

In this essay, I am not concerned with incorrect data models or incorrect code. Incorrect data models usually don't make it all the way to implementation. Unfortunately, incorrect code often does; and when it does, the consequences can be very serious.

But bad data models, as opposed to incorrect ones, do get implemented. And when they do, they can potentially do much more harm than bad code. That's because every piece of code that accesses data in a database does so through the data structures defined by the data model. So if the model is bad, every piece of code is affected by it. A bad piece of code, on the other hand, is just a bad piece of code. No other part of an application system has to compensate for it.

This is a fundamental asymmetry in application systems. Code depends on data structures. Data structures do not depend on code.

So instead of trying to define the difference between good and bad data models, let's look at an example of a bad model, and try to understand what makes it bad.

### **A Data Modeling Vignette.**

Mike and Sally were given four weeks to come up with a data model for their project. The four weeks went by, and they produced what they thought was a pretty good model. According to the project plan, it was then time to look at some of the source systems that would contribute data to the database, and confirm that the model could handle that data.

One of the tables in their model was a Location table. To keep it simple, we will focus on just two attributes of that table. Location number was the primary key of the table, and location name was a non-key attribute. This table is shown in Figure 1.

loc-num (PK)	the identifying number for a location.
loc-name	the name of the location

**Figure 1. A Location Table.**

After doing a little research into the source systems for the location data, Mike came back to the project room, where he met Sally. Their conversation went something like this:

Mike: I just found out that location number isn't unique.

Sally: What do you mean?

Mike: The same location has different numbers depending on the contract that specifies it. It even has different names, depending on contract.

Sally: So what should we do?

Mike: Well, location number can't be the primary key of the Location table anymore, because it's not unique. Location name isn't unique either, so it can't be the primary key. We'll just have to make location number a non-key attribute, and create a system-generated primary key instead.

Sally: OK, let's do it.

So Mike and Sally went off and modified the location table to look like this:

loc-id (PK)	A system-generated unique identifier.
loc-num	A number that refers to a location.
loc-name	a name used to refer to that same location.

**Figure 2. The Revised Location Table / the Location Alias Table.**

From Mike and Sally's perspective, this new table was just a revision of their original Location table. But as we shall see below, the revised table really isn't about locations anymore; it's about aliases for locations.

A sample data representation of the new table is shown in Figure 3:

<i>loc-id (PK)</i>	loc-num	loc-name
2245	B-48	Houston Dock #877
6253	AX-19	Greenfield Park
4047	CWW-721	South Haven Campus, Bldg 5
9113	MGS-22	#877, Houston Dock
.....	.....	.....

**Figure 3. The Revised Location Table / the Location Alias Table: Sample Data.**

### *What's Wrong With This Picture?*

Most of us would agree that Mike and Sally's new table is a bad response to Mike's discovery about the source system data. But why is it a bad response? The new table appears to be fully normalized. Location numbers and names are still being recorded in the database. So what exactly is wrong with this new table?

One thing that is wrong is that the new table is misnamed. It is not a location table because rows of that table no longer stand for locations. Instead, this new table is a table of location *aliases* – the names and numbers by which different contracts refer to locations. So calling this new table the "Location table" will cause confusion for developers, DBAs and users, all of whom will instinctively assume that every location is represented by one row of this table, that a count of rows indicates the total number of locations, and so on. Those assumptions are not true, and that is why it is misleading to call this a location table.

But what's bad about the model is far more than the name of the table. For by replacing the original location table with this one, the model does not represent locations at all any longer. The change did not *add* a table of

location aliases; it *replaced* a table of locations with a table of location aliases. Consequently, the data model provides no way to identify a location any longer. For any specific location, we do not know which rows in the new table are aliases for it. One or more of the aliases shown in Figure 3 may be aliases for the same location, or all four may be aliases for different locations. In fact, looking at the location names, we may suspect that rows 2245 and 9113 are indeed for the same location. But we just don't know. We don't know because we don't have a Location table anymore.

This does not prove that Mike and Sally's revisions were incorrect, however. Perhaps a location table is not required in the model. Perhaps the business perspective is so contract-oriented that contract-specific aliases for locations are all that need to be modeled. This isn't likely, but it is important to understand all the assumptions underlying our critique, so we can then proceed to confirm those assumptions, or discover that we were wrong in making them.

In this case, one indication that locations, and not just location aliases, are important is that there is a location *concept*. Mike used that concept when he said "The *same location* has different numbers ....." Let us assume, then, that conversations with subject matter experts provided additional evidence that locations were within the scope of the subject matter being represented. Does *this* prove that Mike and Sally's model is incorrect? Unfortunately, it does not.

### **The Bad News.**

The bad news about Mike and Sally's revision to their data model is that it might get implemented. In spite of being a bad model, it might not be an incorrect one. Here is one way that could happen.

Suppose that a database was created on the basis of Mike and Sally's model and was then loaded from source systems. Update and query screens were defined against that database, and the system was deployed. Shortly after deployment, however, users quickly discovered that there was no Location table in the database. In response to a change request, the IT department said

that it would be six months before they could fix the system, and add a location table.

So the users came up with an interim solution. They developed a convention to identify all the aliases that really represented the same location. The convention was to use the first five characters of the location name to indicate the real location; all names for the same location would start with the same first five characters. With that convention, users were able to run reports to see all the location numbers and names for the same location, to count locations, and so on.

Next, the users realized that the developers forgot to include location mailing address, which they also needed. Realizing that the IT department would take months to make that change, they created another convention. They kept the convention that location name would begin with a five-character location identifier, and added a convention that location mailing address would follow that identifier, preceded and followed by a dash character, with the alias name itself following the second dash.

The users figured out how to split the location name attribute into its three components on their reports, and how to query by location, by address or part of the address, or by specific alias. In short, they made the system work.

This is how bad models work. Other parts of the application – perhaps the code, perhaps conventions agreed upon by users – compensate for the deficiencies of the model.