Big Data and the Coming Conceptual Model Revolution

By Malcolm Chisholm
APR 10, 2012 5:40am E

Conceptual modeling, or semantic modeling if you like, is a rather nebulous area in data management. There seems to be a lot of agreement that it is needed, some disagreement about what it is, and little understanding of how to do it. Yet I believe we are now at a point where we will be forced to deal with it in a far more serious way than we have in the past.

Definitional Problems

I define a conceptual model as "a model of business information purely as information without any concern as to how it might be stored as data."

To me a conceptual model is not a data model in any sense because it is not part of any effort to design a data storage solution. It is a model that captures information used in a particular area of the business.

Other definitions of "conceptual model" exist. Confusingly, the ANSI/SPARC definition of "conceptual schema" [http://en.wikipedia.org/wiki/ANSI-SPARC_Architecture] is something that describes "... all the data items and relationships between them, together with integrity constraints (later). There is only one conceptual schema per database." This is essentially what is commonly called a "logical data model."

Then there is the "conceptual data model," defined by Tom Haughey [http://erwin.com/expert_blogs/detail/the_conceptual_data_model/] as "a high level or coarse data model which is preliminary in structure, possibly abstract in content and sparse in attributes, that is intended to represent a business area. It is preliminary in structure because it may contain many-to-many relationships."

I do believe that a conceptual data model has a place in data management, as a preliminary to a logical data model. However, it lacks the detail I would expect of a real conceptual model and suffers from being oriented to a data storage design rather than a full description of a business reality.
Data Models and the Relational Paradigm

There is strong evidence that conceptual models are becoming more important today than they have ever been. Essentially, conceptual models are becoming divorced from traditional data models, and the divorce is likely to be a messy one because of the way that data models and data modelers have grown up since the 1970s.

Data modeling as we know it today is inextricably linked with the relational database paradigm - the way in which the columns of database tables are all "related" together. The relational paradigm is so ubiquitous that data modelers do not realize just how much data modeling presupposes it. And the relational paradigm has been enormously successful. It has been tempting to think, therefore, that a logical data model can truly represent the business - to think that a logical data model is the same as a conceptual model.

Enter Big Data

But now things are changing. The success of columnar databases in ultra-large scale data environments has presented a challenge to the relational paradigm. Of course there is enormous hype about big data, but it is also enough of a reality to demand attention. To use the columnar databases successfully you have to unlearn the relational paradigm. I have seen this on a petabyte-scale project I worked on, and it can be ugly. Once the relational paradigm is jettisoned, data modeling as we have known it goes out the window, too. Yet the need to understand what to make of the data in business terms remains. The challenge of managing big data is to distill it into forms that fit the models that business users have of their information requirements - to distill it into conceptual models. Of course it is also true that data models are needed to design a big data dataspace, but these are also unrelational and must come after detailed conceptual models. The reason is that in big data there is no approximation of the conceptual model and logical data model as there can be in the relational paradigm.

And so it was too in the era before relational. ISAM, VSAM, IMS, ADABASE, IDMS and the prerelational data stores could not be designed using ER-based data modeling techniques built on the foundation of the relational paradigm.

And, if truth be told, this is also true when relational databases use generic patterns to hold data. For instance, I recently spent several weeks producing conceptual models for different types of institutional customer housed in a "party model" generic database. My conceptual models bore no resemblance to the design of the relational data store.

What Data Modeling Cannot Do

If we truly model business information in full detail and compare it to what we find in typical data models, there are significant divergences. There are things we need to represent in conceptual models that either are not represented or cannot be represented in traditional data models. These include:
• **Relationships between non-key attributes in an entity.** For example, Total Sales Amount is related to Total Sales Amount Currency by the relationship "is denominated in," but the relationship cannot be expressed in a data model.

• **The use of code tables.** Wherever a code table is used, the physical records it contains represent business concepts that have not been captured and defined in the data model used to design the database in which the code tables are housed. I submit that half or more of the business concepts required for a database can easily exist as code table records and thus be missing from the corresponding data model.

• **Levels of abstraction.** Total Sales Amount as a column in a database represents a piece of business information. But Date-Time of Last Update is metadata about a record. If an entity contains both attributes that are data and attributes that are metadata, I cannot represent them as being at different levels of abstraction. At best I can use devices such as naming conventions, but these are not really satisfactory.

Conceptual models must capture all business concepts and all relevant relationships. If instances of things are also part of the business reality, they must be captured too. Unfortunately, there is no standard methodology and notation to do this. Conceptual models that communicate business reality effectively require some degree of artistic imagination. They are products of analysis, not of design.

Today, big data environments present us with a gap between data storage design and business information that cannot be bridged in a single type of model. The reality is that this problem has always been present to some degree, even during the heyday of the relational paradigm. How the problem will be solved is a different matter, but it will eventually be solved.

*Malcolm Chisholm, Ph.D. has over 25 years of experience in enterprise information management and data management and has worked in a wide range of sectors. He specializes in setting up and developing enterprise information management units, master data management, and business rules. His experience includes the financial, manufacturing, government, and pharmaceutical industries. He is the author of [How to Build a Business Rules Engine](http://www.bizrulesengine.com) and [Managing Reference Data in Enterprise Databases](http://www.refdataportal.com) and [Definition in Information Management](http://www.data-definition.com). He writes numerous articles and is a frequent presenter on these topics at industry events. Chisholm runs the websites http://www.bizrulesengine.com, http://www.refdataportal.com and http://www.data-definition.com. Chisholm is the winner of the 2011 DAMA International Achievement Award.*
The Conceptual Data Model

By Tom Haughey on March 4, 2011

A conceptual data model is a high level or coarse data model which is preliminary in structure, possibly abstract in content and sparse in attributes, that is intended to represent a business area. It is preliminary in structure because it may contain many-to-many relationships. It is abstract in content because it may need to suppress some details so as to subsume different subtypes. It is sparsely attributed because it will usually contain only major business attributes. Because of these characteristics, you cannot implement a conceptual data model. An example is shown at the end of this blog.

Given these characteristics, why in the world would we have one? There are three main reasons:

- **For management presentations.** If you have to justify or explain a project to management, you do not have forever. Management has a short attention span. This is not a value judgment about management. If you are presenting to senior, or any non-technical management, you need to get to the point in (say) 15 minutes or so. You have to be able to explain the issue to them clearly and quickly. A conceptual data model is ideal for this purpose. For this reason, the conceptual data model should fit on a single page.

- **For planning.** A project that is in the planning stages does not need detail yet. You need only know what overall data, and sometimes only what types of data, are contained in the business area. A mass of detail is not necessary for this. The conceptual data model serves this purpose.

- **To start top-down modeling.** There are two circumstances for which a conceptual data model is necessary: new development and major maintenance. New development is development from scratch. Only about 20% of software projects are new development. It is dangerous and cumbersome to do new development only bottom-up. In major maintenance substantial new requirements are added to an existing system. Such projects will start bottom-up when reverse engineering the existing database. The conceptual data model will be necessary to understand in what way to introduce the new requirements.

Let us expand our thoughts on bottom-up in data modeling. Use-case modeling is very popular these days. Use-cases are a simplified form of process modeling and an enriched form of process definition. It is helpful to employ use-cases to expand a data model. BUT the data model should not rely exclusively on use-cases. All data modeling should always be a combination of top-down, bottom-up and middle-out. New development starts top-down. Maintenance starts bottom-up. Developing a data model exclusively from use-cases is essentially doing a data model bottom-up. It restricts the kind of creative thinking that should naturally go into a data model. The conceptual data model is used to start the top-down
process. The use-cases provide the bottom-up perspective. Working with the data model in and of itself, exploring business needs, and then expanding the data model from within is the middle-out part of the process. It is where the real creativity occurs. The data modeler and use-case modeler should be simultaneously exposed to requirements from the business experts. Then, individually, each develops its own unique deliverable from these; the data model in one case, the use-case in the other. And then these two models should be concurrently interacted and balanced one against the other. A future blog will discuss model interaction.

Caveat
The term “conceptual” is not an absolute term with one usage across the IT industry. For example, any methodology or product based on the French Merise methodology will define “conceptual” differently than in this blog. In the Merise methodology, the term “conceptual” refers to the fully detailed, business oriented data model that is independent of optimization and technology. In the US, the term “logical” data model is generally used to describe such a model.