Data Modeling in the Age of Big Data

Pete Stiglich is a principal at Clarity Solution Group.
pstiglich@clarity-us.com

Pete Stiglich

Abstract

With big data adoption accelerating and strong interest in NoSQL technologies (such as Hadoop and Cassandra) and the flexible schemas they offer, the need for data modeling is sometimes questioned. However, data in NoSQL may still require some structure so data scientists can mine it effectively, even though NoSQL does have powerful mechanisms for working with unstructured data to extract structured data.

Whenever data has some structure, a data model can be useful for describing it and fostering communication among data scientists, developers, data analysts, and others. The need for conceptual (business/domain) and logical data modeling doesn't change significantly, although physical data modeling practices may need to be adjusted.

The concept of a data lake, where all your source system data is stored on the massively parallel processing (MPP) cluster or grid, requires effective metadata management (and possibly data modeling) to help you understand the data when source system documentation is inadequate. Determining how a data modeling component fits into your big data projects requires collaboration among management, project managers, and application and data architects.

Data Modeling in the Age of Big Data

With the advent of NoSQL databases and big data platforms such as Hadoop, Cassandra, and MongoDB, the need for formalized data modeling is sometimes questioned. Some believe that formalized data modeling (in the sense of defining data structures, not in the sense of data mining/statistical models) can and should be circumvented in NoSQL and big data environments. After all, these people argue, data modeling slows down development and models aren't necessary given many of the "schema-less" features of these new platforms. However, whenever you are designing data structures,

you are modeling data, whether formally (e.g., using diagrams) or informally (e.g., data definition language).

In this article, I will describe how data modeling is still highly relevant in the big data world. I will explain how data modeling will continue to evolve, as well as how data modeling for NoSQL big data technologies changes or has similarities with traditional data modeling. This will include:

- Determining when and how data modelers should be involved in big data projects
- Differences and similarities between relational database management systems (RDBMSs) and NoSQL (because this affects data modeling)
- Types of data models to develop
- How data modeling tools can be used in a NoSQL environment

What Does "Schema-less" Mean?

Even when you work with unstructured data (video, audio, freeform text), your data must have some structure to provide meaning or be useful. Unstructured data usually has associated metadata (e.g., tags, creation or modification dates, Dublin Core elements) to provide context and so that the metadata and the data together have a structure. Many of the new big data technologies leverage non-relational data stores based on key-value, wide-column, graph, or other data representation paradigms.

Schema-less typically refers to the absence of predefined physical schema that are typical of relational databases—for example, in system catalogs where tables, columns, indexes, views, etc., are pre-defined and often relatively hard to change. However, unless you are doing something similar to using MapReduce to create key-value pairs by interrogation of a file (e.g., to get word counts from unstructured text), your data typically has some kind of predefined structure. For example:

- Pig and Hive have similarities with RDBMS tables, such as tables, columns, data types, and partitioning. There are also differences, of course (such as no foreign key constraints, and no unique constraints).
- HBase and Cassandra have tables (that are completely denormalized because joins aren't possible natively) and column families that are predefined, although the actual columns (aka column qualifiers) can be completely variable (e.g., one record can have 10 columns and another record can have a million completely distinct columns).
- MongoDB stores data in JSON documents, which have a structure similar to XML.
- Semantic models, although perhaps using an RDF triple store for persisting, have taxonomies and ontologies that require modeling expertise to develop. Making sense of unstructured data often requires such taxonomies and ontologies, e.g., to categorize unstructured data such as documents.
- The need for maintaining a "system catalog" for Hadoop-based tables has been recognized; HCatalog has been developed to help keep track of these catalogs.

In the big data world, your data often has a schema or structure, although there can be a high degree of flexibility.

Role of Data Modelers in Big Data

Given the flexible schemas of many big data technologies, where (and when) does the data modeler fit in? The answer becomes more difficult with NoSQL environments, and involvement will vary by type of project, industry, and sensitivity or criticality of the data (e.g., financial, data required for compliance, PHI data). An Internet company may require less formal modeling than a healthcare provider, but multiple development paradigms may be needed even within a single company. The more critical the data, the more quality required, the more structured the data is, or the more it is exposed to end users or customers (via a BI tool, for example), typi-

cally the greater the level of modeling formalism required. However, there is no one-size-fits-all approach.

Data architects and modelers should generally be a part of most big data development teams for:

- Modeling business objects and relationships as a precursor to a design phase and to help data scientists understand how to tie the data together from a business perspective
- Translating business data requirements into target data design (logical and/or physical) depending upon project type
- Providing semantic disambiguation and helping data scientists and developers make sense of the data (e.g., as a data archaeologist)
- Modeling source data (e.g., reverse engineering) and developing artifacts to help formalize communication about what source data is contained in the data lake and the business rules associated with the use of individual data components
- Identifying, collecting, and providing source metadata (e.g., source models, definitions, data dictionary)
- Enabling alignment with data governance, as applicable

Management, project leaders, and architects (application and data) need to collaborate to determine the best approach for roles and responsibilities on a particular big data project.

Application development and data modeling require different skill sets and have different perspectives, and projects benefit from reconciling diverse opinions. When application developers design a database without proper training and experience, they are generally focused on the short-term needs of the specific tactical application development goals. They may have limited views toward enterprise use of data/information, and will likely create a database that meets the needs of the immediate application, although over time the system may:

- Not align with corporate/enterprise goals
- Degrade data quality
- Become difficult to integrate with other applications and databases
- Require extensive rework
- Perform or scale poorly
- Fail to align with data governance and stewardship decisions

Consider the case of application developers (and sometimes inexperienced data modelers) who might proceed immediately to physical data modeling, bypassing conceptual and logical modeling, to create the schema. As a result, business and functional relationships might be modeled incorrectly (e.g., a many-to-many relationship—from a business perspective—instead of a one-to-many). The results can be missing data, data duplication required to fit the data into the model, mischaracterization of data, time spent by data scientists waiting for clean data (and they're not inexpensive), and loss of confidence in the ability to deliver quality information.

Data scientists need a sandbox where they are free to create their own tables, manipulate data, and formulate and test hypotheses. In this case, the data modeler's involvement should be to support the data scientist's understanding of the data. Similarly, application developers may need similar flexibility to prototype or rapidly develop solutions. For applications destined to be put into production, the data model provides an architectural foundation that supports the overall organizational goals and standards and should be a key component.

As an example, when developing a reporting environment, a data modeler would design a model (e.g., a dimensional model with facts and dimensions) for an RDBMS. The data modeler could develop a similar dimensional model for implementation on Hive. Where there are differences, data modelers must learn the new NoSQL technologies to effectively support big data initiatives, much as they transitioned from a traditional RDBMS to data warehousing (DW) appliances.

Data architects and modelers need to be concerned about enabling integration and effective data management without slowing down the development process. They must work to standardize the naming, definitions (business description, logical data type, length, etc.), and relationships of the data whenever possible to prevent future integration nightmares and to manage the data effectively. Different technologies have different ways of physically naming items; data modelers should ensure that the same concepts/business terminology is leveraged. For example, if a customer is called a customer and not an account in the enterprise data model, the same term should be used in new data stores. This practice reduces future integration problems, even if the physical representation of the name needs to be different due to technical constraints.

Data is an enterprise asset that must be identified (including resolving and tracking synonyms and homonyms); understood and defined; tracked for effective stewardship, data lineage, and impact analysis (if you don't know where your assets are, you're not managing assets well) in a metadata repository or business glossary; and have its quality, security, and usage identified and measured. Data architects and modelers help to ensure the data is aligned with enterprise standards whenever possible and have an eye on the bigger picture—looking beyond the current project scope.

Data architects and modelers understand how to translate business requirements into the appropriate design for a data store. Indeed, conceptual and logical data models can and should inform the choice of database platform (relational, wide column, graph, etc.). Not every application works equally well on all platforms. For example, if a generic data model type such as entity attribute value (EAV) seems indicated—where the entity and attribute names are determined at the time of insert rather than having a table per entity each with predefined columns—and the volume is significant, a wide column store such as HBase or Cassandra might be better. If there are many relationships in the data (such as with social networking), perhaps a graph data store is called for.

How Data Modeling Changes with Big Data

The standard modeling processes are the same. The first data model to develop is a conceptual data model (CDM), which identifies key business objects and their relationships. Even when the system must handle unstructured data, there are still business objects and relationships that must be identified, even if just for the metadata providing context to the unstructured data. The CDM (using ERD, UML Class Diagram, ORM, or other modeling notation) provides a framework for an information system. It is roughly equivalent to an architect's conceptual design of a building.

Of course, we would never allow someone to build a house without seeing a picture of it beforehand—yet, strangely, many in IT don't have a problem with going straight to schema development. This would be like a builder drawing a layout of the electrical or plumbing system without knowing what the building is going to look like. Obviously, conceptual data modeling is still required for any information system intended for production—the CDM is the framework for an information system. Nothing is worse than building a data store where the data and relationships don't align with business requirements.

Some may argue that in a big data environment you should just put all source data onto your cluster and let the data scientists hammer away and manipulate the data as desired. There is definitely real value to enabling this scenario, and you don't want your data scientists waiting for all the data to get integrated into the EDW before they can start data mining. Keep in mind two key points:

- Data scientists must understand the business and, of course, the data and relationships. The conceptual

data model will allow the data scientist to better understand the business objects, how they relate, and the cardinality of the relationships. This will help data scientists understand how to navigate the data and identify data quality issues that may skew results.

- Integration and data quality assurance are difficult and time consuming. Most of a data scientist's time is spent gathering, integrating, and cleansing data before even starting analysis. Did I mention that data scientists are expensive? Initially, all the data can be stored in the data lake and models and other metadata collected and provided to make sense of the data. Eventually, more data can be integrated, cleansed, structured, and pulled into the EDW or retained in the big data environment as appropriate. By cleansing, standardizing, and integrating this data once and persisting it, it will be available for future analyses that can be performed more rapidly. Providing a semantic layer "schema on read" can help make column names more business oriented to improve understandability.

One notion about data science is that it is important to be able to "fail fast"—to minimize the impact and risk when a data mining algorithm doesn't produce usable or valuable results. Storing all the data in a big data environment and allowing data scientists to analyze and manipulate it is a worthwhile goal. However, without the enterprise data model (which may include conceptual and logical models), source data models and dictionaries, mappings from the source data to the enterprise data model, and data profiling results from the source data, your data scientists will be less effective. When data from multiple systems must be integrated for analysis, more time must be allocated, even if all the data is stored on the big data platform.

The Logical Data Model

The next model that should be created in most cases is the logical data model (LDM), though it is not always required. For example, if you are developing a graph database, you might convert your CDM into an ontology (e.g., using RDF/OWL) as the foundation for your semantic model/graph database.

The conceptual data model should only use business terminology; abstractions can cause confusion for business managers, analysts, and data stewards who need to review and approve the model. The CDM might be lightly "attributized"—some attributes (such as sales region, country, or e-mail address) may be considered a business object and have relationships that should be expressed, and other attributes (e.g., first name, sale date, or sale amount) would probably not be considered a discrete entity.

An LDM often includes abstractions of entities, attributes, and relationships to increase reuse. For example, in a CDM, entities such as patient, professional provider, employee, and representative might be abstracted into a generic entity called "person" in the LDM so you're not storing duplicate name, demographic, and contact information. An LDM often begins to represent a data structure that can be implemented in a data store, and thus is the start of your actual design. The CDM is not part of design. Instead, it describes the business from a business object perspective and is performed in the requirements gathering phase.

A critical component in the LDM is the identification of keys, primarily natural keys. Although the NoSQL technology might not employ indices (unique or otherwise) or have key constraints, it is still very important to understand what attribute(s) uniquely identify a record. Even when partitioning relational tables, it can be difficult to enable a unique index because the partition key might need to be a subset of the primary key (and the natural key is not always the primary key, such as in the case of surrogate keys). In any case, in the LDM you must be sure to identify natural keys (as either primary or alternate keys) so that the granularity and meaning of the entity can be more accurately understood and serve as an aid for the use of and communication about the data.

An enterprise logical data model is constructed based on a normalized pattern independent of application (even enterprise applications such as MDM or EDW) so that entities can be defined at the atomic level to enable future data integration. An enterprise logical data model may retain many-to-many and subtype relationships (which

aren't implementable without transformation, such as associative entities to resolve an M:N relationship) in order to allow the applications to transform these as appropriate to the application.

An application logical data model may be normalized or denormalized as applicable. For an application leveraging a NoSQL wide column store such as HBase or Cassandra that doesn't allow joins natively, the application LDM might be a single denormalized table with the row key and commonly used columns or column types defined. Alternatively, column families could be defined as separate logical entities when multiple column families are envisioned to aid communication and understanding.

At this time, I am unaware of any support in commonly used data modeling tools for NoSQL constructs such as column families, so the data modeler will need to adjust usage of the modeling tool as necessary. Column families may also be defined based on physical characteristics such as encoding or memory pinning. Column families defined to support performance or other physical characteristics should be defined in the physical data model or in the table schema, not in the LDM.

The Physical Data Model

The last step in the data modeling process is creating the physical data model (PDM) that the schema will be based on and where physical characteristics of the tables (encoding, format, partitioning) are determined. Unfortunately, the traditional data modeling tools provide little support for NoSQL, so forward engineering the PDM to create DDL appears to be out of the picture for the time being. The NoSQL DDL will probably need to be developed outside of the modeling tool manually or using a tool such as AquaData Studio. Developing a PDM becomes less important than the CDM and LDM, but may still be useful for demonstrating denormalization (e.g., HBase or Cassandra) for physical naming (instead of using business names in the CDM and LDM), and for metadata retention. Even if DDL can't be generated, describing the physical data structures and showing how they relate can still yield benefits.

Summary

The age of NoSQL big data is clearly pushing a paradigm shift away from the familiar relational database world. However, much remains the same about the need to model and manage data. Effectively translating business requirements into data structures intended for a production environment still requires the skills of a data modeler.

Data governance and stewardship decisions and standards need to be followed in NoSQL data stores because data is an enterprise asset. For data scientists to answer the really tough questions, they must understand what the data means and how it ties together. Data models and modelers provide invaluable assistance to the data scientists. Data modelers need to learn NoSQL technologies so they can remain relevant and provide the best service possible. Data modelers and application developers both need to be adaptable and work well together to build optimal NoSQL solutions. Data modeling tool vendors need to support NoSQL constructs because NoSQL big data technology adoption is accelerating. ■