Lean data modeling:
The art of mitigating risk for changing requirements

Paul Delgman & Tom Breur
July 2016

Introduction
Analytics serves to support business goals. Development teams that work on data driven analytics solutions, need to understand what business objectives their product should support. How should decisions based on this information product contribute to the bottom line? Maybe by growing the customer base, more cross-sell to existing customers, less attrition, etc. To appreciate these business objectives, there will probably be some meetings called “requirement gathering.” And after some strenuous sessions where every question you can think of will be asked, data professionals will present a high level blueprint that describes the proposed solution.

Just when you think you covered all the bases, another big challenge arises. As time goes by, things begin to change. As much as we always want and love to get the requirements “just right” the first time around, the fact is we don’t. Such is life. Besides death and tax, change is the only other certainty. And even when we *do* get the requirements right for the current situation (at least at first), they will change over time as circumstances change. Suffice it to say, changing requirements are a fact of life. Not taking such anticipated changes into account is naive and downright poor (amateurish) design.

The intricate ways in which (unavoidable) changes in requirements can impact data models, reporting and analytics is the core messages of this article. When a team of developers sits down with business stakeholders, the *process* of modeling the data helps both parties understand options and constraints. And unless you both understand how possible future changes in requirements and the data model will impact your design, you leave it to chance when and how you will find out.

From experience we have learned that sometimes you find out “the hard way” how expensive those consequences might be. Exactly that scenario is what we suggest you try to avoid by making the data modeling process an inclusive and collaborative process. You want all relevant stakeholders involved and contributing. The output from this data modeling process (sometimes referred to as “modelstorming”), the tangible data model, needs to be jointly owned by all stakeholders in the development process.

This article explores some of the most impactful decisions in data modeling exercises to help non-data experts understand the potential impact of their questions, statements or requests on the optimal choice of data model(s).

The data modeling process
Data models matter. A lot. The data model is a key enabler for any functionality that requires data. The data model, although largely invisible, will determine how you are able to leverage

your data, and maybe just as important: also what *isn't* possible. In fact, it isn't only the *outcome* of these efforts (the tangible data model) that matters, the *activity* of modeling itself is instrumental in clarifying requirements, and surfacing what the best way is to ensure these are met now, **and** in the future.

The hard part, and where data modelers have a chance to shine, is that you never build something "for once and for all" - instead your solution *will* need to change (ironically called "maintenance"). Good design (choosing "the right" data model) ensures these so-called maintenance costs are and *stay* bearable.

Data models are the foundation for any technical solution involving data. They are required to accommodate today's as well as tomorrow's business needs. They explicate how the solution works, or is supposed to work. As such, data models are an artifact that is essential to keep maintenance costs in check. The pace of change in businesses and markets means data modelers face a huge challenge keeping the costs of change in check.

One of the challenges starts at the outset, with the data modeling exercise *itself*, to ensure you make it engaging and entertaining, and not make it too "technical" so as to alienate essential business stakeholders from this process. The last thing you want is users (business stakeholders) rolling their eyes, assuming the data model is just some technical artifact, mentally retreating from the effort. For some IT folks it may never get technical enough. Yet for others, mumbo jumbo jargon is not-so-entertaining at best, or downright intimidating at worst. Data modeling should be fun, for all participants! And unless we make it so, it becomes a technical elitist, isolated, and hence dysfunctional activity. Fortunately more and more people are working to improve the data modeling process. Lawrence Corr, for example, has created an easily understandable approach (called BEAM) to model data in star schema's, and make this process accessible and understandable, also for non-technical stakeholders.

The data model absolutely *has* to be created with a broad perspective of all potential future changes in mind. Consequently, data models always need to be designed and built with maximum flexibility, so that (nearly) all predicted *and unpredicted* changes can be adopted at the minimal possible cost.

This is the main reason why data models need to be created *together*, so that they serve to clarify requirements ("scoping"). Deliberate choices need to be made that can have a dramatic impact on maintenance costs. The reason why the majority of IT costs are attributed to "maintenance" is because changes will happen over the lifecycle of a solution, and the data modeling phase at the outset of a project is the time to ensure these costs will stay in check. In close collaboration you can anticipate where and when change is to be expected, and then choose "the optimal" data modeling solution accordingly.

Early development decisions

Another big challenge (risk) is that some important foundational decisions need to be taken fairly early on in the development process. This happens regardless of whether you are working in the context of an Agile team, or not. Different data modeling paradigms and storage architectures each have their own pros and cons. Commonly used data modeling techniques are 3NF ("Normalized"), Dimensional modeling, Data Vault, Anchor modeling, and there are more. Data storage has been mostly relational for the last few decades, but now we also see NoSQL and NewSQL solutions, and these seem to be growing in popularity. Mostly the new platforms allow so-called "Schema on read", which means that the data modeling "only" needs to be completed by the time you read the data, not (yet) when you store it. Each and everyone of these choices will imply constraints and limitations to the eventual solution.

Depending on the requirements (!), design decisions need to be made under considerable uncertainty. Some of these, like choosing a storage platform, or choosing between "traditional" relational (SQL) solutions or the newer NoSQL solutions, will be difficult and (very) expensive to change later on! These are the kind of high-risk decisions that you simply cannot postpone if delivering some functionality ("business value") early on in the process is your preferred mode of working - and it certainly is ours!

Even working in an Agile team where applications grow incrementally, some decisions might need a bit more consideration than others. An ounce of prevention can save a pound of regret: a poor choice early on might cause considerable technical debt later on. Understanding which decisions have the biggest impact on your future development pace can make the difference between successful or death march projects. Here you will need to rely heavily on seasoned data experts, because they are likely the only ones who have the experience and knowledge to make these judgments, both from a business process perspective as well as from a technology perspective.

Critical data modeling topics

Data models can always be created in multiple renditions. Although this may appear counter intuitive, the same underlying source data can be stored and organized in several ways, and all of these representations can be "correct" or "valid" from a data modeling perspective. This is to say that from a purely mathematical standpoint, there are multiple ways of accurately representing the underlying primary business process. The differences come in how cumbersome or costly these solutions are to build, to use, and to change.

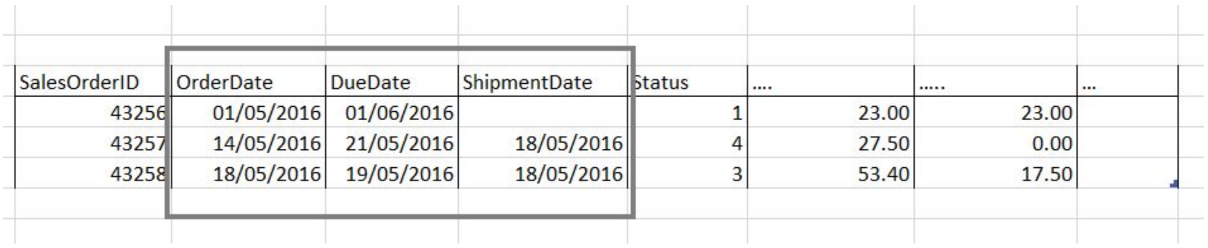
A core premise of our paper is that depending on the initial requirements for the project, the choice for modeling data in one way or the other will have a material impact on the ease with which you can deliver information. How quick you may be able to provide some initial reports, but also how difficult or expensive it may be to make changes later on. The deliberate choice for rendering the underlying source data in a particular way can have considerable impact on how cumbersome and time-consuming it will be to make changes later on - a major cost factor! Unless all parties involved are properly informed and educated about these consequences (and the corresponding upfront investments required), the requirements phase is incomplete at best, or downright dysfunctional if you get unlucky. We've lived through painful memories where business stakeholders learn "the hard way" that some seemingly minor changes down the line turn out to be very costly. This kind of disappointment with budget owners can dramatically chip away at trust and confidence. We, as architects, need to own up to these scenarios and lead discussions to surface such trade-offs.

This is an example where an early design decision, in this case deciding to model the data in a certain way, can have considerable impact (positive or negative) later on in the project. The “right” data model needs to accommodate not only the current business requirements, but must also be flexible enough to adapt to possible (plausible) future changes. Data modeling is the art of choosing to store data in a particular way.

In this paper we’ve identified three critical requirement area’s that are critical to the data modeling exercise. These area’s need to be explored extensively and the effects of choices need to be understood by all involved. Incorrect assumptions or changing requirements in any of these areas potentially cause expensive changes that might require months to implement at great expense .

Time
Time can be one of the most confusing areas in data management. To the layperson, everything starts with a single timeline, the calendar, which is obvious and easy to understand. Something happens on a specific day, at a specific time. But there are some hidden challenges with regards to this topic “time.”

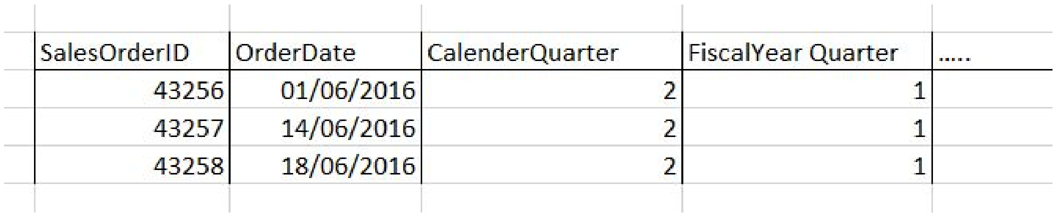
The first is having multiple timelines, when for a specific event there are two timelines available. For instance “orders” can have a “order issue date” (the actual date-time the order has been placed) and a “order shipment date” (the date-time the order has been processed & shipped). Depending on the requirements one of these might be the “leading” timeline, or both could be required.



SalesOrderID	OrderDate	DueDate	ShipmentDate	Status	....	.....	...
43256	01/05/2016	01/06/2016		1	23.00	23.00	
43257	14/05/2016	21/05/2016	18/05/2016	4	27.50	0.00	
43258	18/05/2016	19/05/2016	18/05/2016	3	53.40	17.50	

Figure 1: multiple timelines

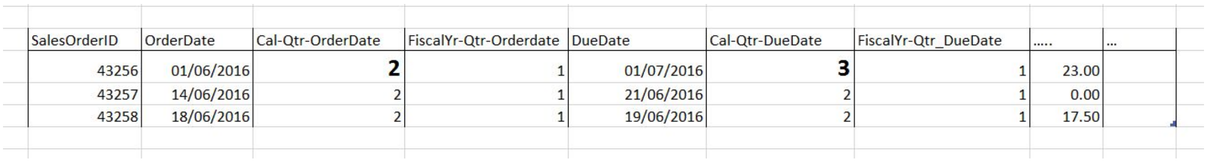
A second challenge with timelines is “aggregating.” From a calendar perspective again, this is easy: for any event we know to which month, quarter and year it belongs. But what if your company would use a different accounting calendar? Where July in calendar terms is 3rd quarter, but from the perspective of the accounting calendar it might be your 1st quarter? And what if you have seasonal product lines, and their “quarters” (cyclical patterns) don’t start at the first of the month, but at the 20th? We have seen companies with over four different timelines used throughout the company, which brings additional challenges to anyone working with data.



SalesOrderID	OrderDate	CalenderQuarter	FiscalYear Quarter	.....
43256	01/06/2016	2	1	
43257	14/06/2016	2	1	
43258	18/06/2016	2	1	

Figure 2: Multiple aggregation levels (Quarters)

If we combine these two challenges (multiple timelines and multiple aggregation levels) because we inferred during the requirements phase that we want to be able to analyze against all timelines and aggregation levels, we get a multiplication of additional information, that will show differences:



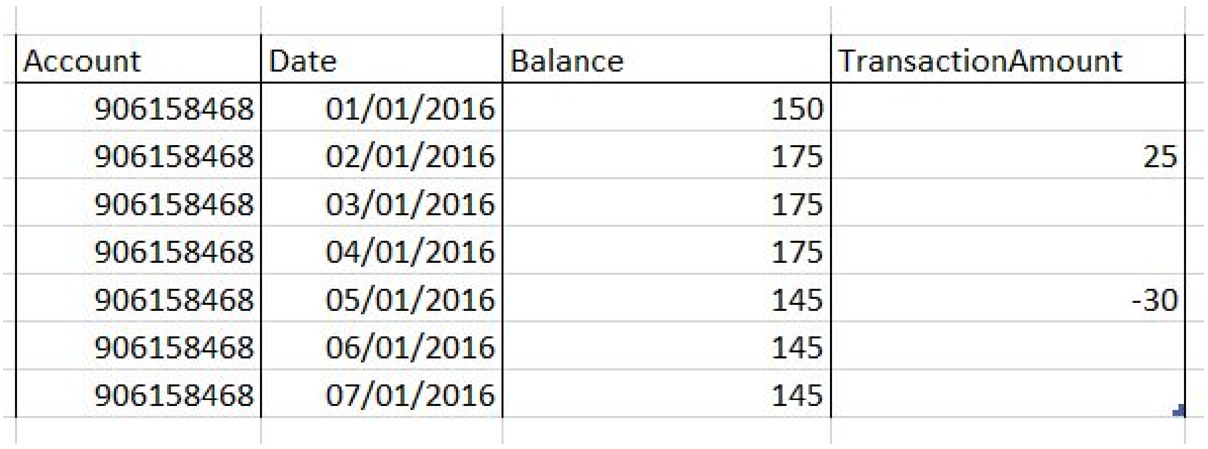
SalesOrderID	OrderDate	Cal-Qtr-OrderDate	FiscalYr-Qtr-Orderdate	DueDate	Cal-Qtr-DueDate	FiscalYr-Qtr_DueDate	.....	...
43256	01/06/2016	**2**	1	01/07/2016	**3**	1	23.00	
43257	14/06/2016	2	1	21/06/2016	2	1	0.00	
43258	18/06/2016	2	1	19/06/2016	2	1	17.50	

Figure 3: multiplication of complexity by combining multiple timelines with multiple aggregations

These examples show how requirements on multiple time lines can complicate data models. These more elaborate data models also require “old” business questions like: “how much did we do in Q3?” to be more specific, e.g.: “How much did we do in Q3, according to our fiscal calendar, based on Due dates?”.

The third and most complex challenge is time-traveling, which is looking at the circumstances surrounding an event at a specific point in time. For example: we’d like to know the balance of our account at january 1st of 2015.

From a data perspective there are two ways to answer this question. The first option is to store data for every single day, so we have only one timeline (the date itself) and we can conveniently select the record pertaining to the required date.



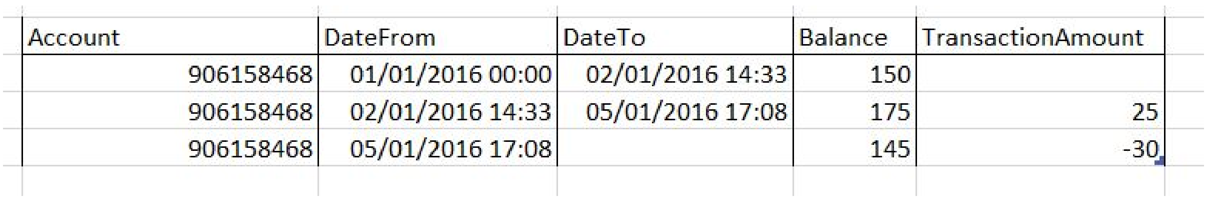
Account	Date	Balance	TransactionAmount
906158468	01/01/2016	150	
906158468	02/01/2016	175	25
906158468	03/01/2016	175	
906158468	04/01/2016	175	
906158468	05/01/2016	145	-30
906158468	06/01/2016	145	
906158468	07/01/2016	145	

Figure 4: storing records for every date

Although this option appears easy, there are some disadvantages to it: the volume of data explodes, a lot of storage is wasted on redundant records (since not every balance changes every day) and you're limited to the grain of data you've selected. What if we want to know our balance per 2nd of January at 13:45 CET from the example above? At what exact time did this 25 get added to the account?.

Choosing a finer grain (say, a separate record for every hour) will make your data volumes grow by a factor 24, and this will therefore create both additional storage (more disks required, that come at a cost) and performance challenges (a faster machine is required to sift through larger volumes of data). No free lunch.

The second option that is sometimes used to enable time traveling is using a timeline within every record. Each record holds both a date/time from which it's valid, and a date/time when it's no longer valid anymore. This means that your volume of data only grows when it needs to: when there is a change in account balance. For storage, performance, and flexibility perspectives this has benefits over the first option, but a major disadvantage is that it becomes a little harder (both more knowledge required as well as a higher risk of errors) to retrieve the right information from your raw data.



Account	DateFrom	DateTo	Balance	TransactionAmount
906158468	01/01/2016 00:00	02/01/2016 14:33	150	
906158468	02/01/2016 14:33	05/01/2016 17:08	175	25
906158468	05/01/2016 17:08		145	-30

Figure 5: introducing timelines at a row level

The impact of changing requirements or faulty assumptions while creating the data model on timelines can differ greatly. A change in the aggregation of a single timeline will be fairly easy to manage, while requiring time-travelling when this is not catered to in the basic data model might throw you back considerably. Unless you get your requirements exactly "right", which is an illusion, you might just run into some unpleasant surprises!

Uniqueness & relationships

One of the central concepts of any type of data are the relationships and uniqueness of records. Working with data we need to understand how to interpret a single event (some activity that has taken place in the real world) or entity (tangible objects in the business world) and how this relates to other data. Even when you're working with unstructured data, it is still crucial to understand how this data relates to other data, in order to enable analysis and create actionable data.

For uniqueness it is essential to understand how the data should look when we're dealing with a single, unique event or entity. Usually we start this exercise with profiling: analyzing available data in combination with business assumptions, and most of the time this leads to a pretty accurate understanding of what is unique. However, more often than not, after some time passes, data models or downstream analysis break down because of unexpected duplicates entering the data. These so-called collisions of data records may seem like minor, infrequent occurrences, but they can wreak havoc and cause considerable rework. Also, although these may seem like tiny amounts of data, the underlying process still needs to be understood. Some significant business events (like fraud, for instance) invariably manifest themselves at first as “outliers”: records that are discarded because they don’t fit the commonly known patterns. Until they start weighing on the bottom-line...

As an example, let’s explore the concept of a “customer”, starting with the deceptively simple question: how many customers do we have? “Simple” questions like these have often proven elusive, in our experience. Suppose our customer is a person, how can we then uniquely identify this person. Will we do so by referring to his or her name?

After profiling the available data, we conclude that it would be risky to rely on names as our unique identifier. Two people can have the same name. How big is that risk? In the U.S. census of 2000[1], for instance, there were 46,664 people named John Smith. There were 1,015 people named James Bond, 107 people named Harry Potter, 454 people named George Bush, and 33 people named Emily Dickinson.

So what if we were to combine name with Date of Birth? Might that work? This leads to a clear improvement, but still there is some risk of duplication. In a group of people as small as about 50-60, statistically you would already expect there to be a 99% chance of at least two people having the same Date of Birth[2].

This discussion could carry on for a while, even to a point where the data we would need to do further profiling doesn’t exist or its availability is restricted by privacy regulations. But in any case, either to meet database constraints, our analytical goals, or even to accommodate accountancy regulations, we need to somehow be able to identify unique customers (or website visitors, or suppliers, etc.) with a prudent degree of certainty. One of the common solutions for this problem is to identify people uniquely by some random ID or an account number (or something similar), which appears safe, at least for the time being. Obviously this solution does not cater perfectly to the same customer applying twice with slightly different personal information, and there are other problems associated with this pragmatic approach. What if your company would acquire some other company, and their accounts would happen to contain the same numbers…? None of these scenarios may materialize (hopefully), but unless you consider them, the costs when disaster strikes are an ugly surprise.

The same logic identifying uniqueness for a person or customer holds for events like transactions and orders. Probably an entire whitepaper can (and should) be written on creating unique events or entities. The basic understanding we need to be aware of is that determining uniqueness is genuinely hard, no solution will hold up with 100% certainty, and even if we can identify unique values for now, that might change tomorrow. What is also important to realize is that although “uniqueness” may seem a purely technical database problem, it has very real and tangible business consequences that can materially affect the bottom-line. These are not the kind of decisions you can leave to IT alone.

---

[1] http://howmanyofme.com/
[2] https://en.wikipedia.org/wiki/Birthday_problem

In some cases, changes in uniqueness can have a very limited impact on the data model when they occur in isolation. But when these so-called record collisions inflict changes in relationships or granularity of the data, the impact typically is a lot bigger.

Relationships can be impacted in multiple ways. The first and most simple one is when the unique identifier is used in other data sets too, which would imply those other data sets or the relation to these data set needs to be altered. The second, more complex change, is when the uniqueness actually changes the relationship itself.

In the example earlier we stated that an account number could be used as an identifier. But what if our product team introduces family accounts where more than one person can own a single (joint) account, or if a single person can have multiple accounts. Working with data relationships we identify three types of relations: one-to-one (every person has only one account), one-to-many (one person can have multiple accounts or one account can have multiple owners) or many-to-many (there's no limit on either side).

A change in relationship might be devastating for your downstream analytics, and could require redesign for large parts of the data model.

Role switching between Facts and Dimensions
Regardless of whether you are dealing with big data, real time analytics, or classic BI, at some point your data will get analyzed from two or more perspectives. In dimensional modeling these are called Facts and Dimensions, but specifically Facts also go by other names as measures (e.g.: Tableau), values (e.g.: PowerBI) or expressions (e.g.: Qlik).

Facts are typically the "content" in your data (*what* you are measuring or reporting on). Usually these are the values that you want to analyze, and therefore require to aggregate (eg: calculations, counts, etc..). Dimensions are typically your column headers in a spreadsheet (what comes after the word "per" when you describe the findings). They are the perspectives or aggregations levels you're using against your facts. For instance, you may sum ("aggregate") the sales, your fact, belonging to a particular region, your dimension. Usually Facts contain far more unique values than Dimensions. Examples of often used Dimensions are location, date or customer(-segment), etc.

For data models, specifically in relational databases, it can be cumbersome if the role of an attribute changes, e.g. when a Fact also serves as a Dimension, depending on the analysis you are trying to perform. For instance: when we want to know the total Order Value over 2015, your measure is the Order Value and the year (calendar) is the Dimension, the aggregation level. Now let's suppose we make our query a little more complicated and we would like to learn the total order value over 2015, categorized per Order Value (how many orders were less than $10.000, and how many more?), then the Order Value suddenly also becomes an aggregation level. The same attribute switches roles from being a Fact to becoming a Dimension: the cutoff value $10.000 becomes a header in the sales report. Most modern (BI) tools can handle this to some extent, but ideally it needs to be handled within the data model to ensure superior performance and usability. This scenario where a Fact (Order Value) can also be used as a Dimension (number of orders with value > $10.000), is a great example of where insufficient clarity with regards to initial requirements can cause painful and costly rework later on.

From the perspective of a report user, it may seem counter intuitive that adding a “small” request (“Can we have Order Value in the column headers, please?”) could cause database developers to frown and wreck their brains! “Why is this so hard? The order value is already in there!” Yes, indeed, the order value is readily available, but depending on how it is represented in the underlying data model this might be a fairly small and simple request, or it can be a brain teaser that the Data team needs to find some nifty solution for. These types of discussions can cause “classic” friction around changing requirements: business users may have the impression they are submitting a small request, and the database designers are wondering why on earth the end-users didn’t ask for this from the start...

An interesting phenomenon that we have observed many times in data projects is that as soon as data become available, the change requests start coming in. “Can you please add this?” “Can you change that?”, etc. Many users are “hungry for data” - they are starving for information! We consider this a sign of success and not necessarily a failure in requirements gathering, since for sure the data are being used now, or no one would bother to ask for any changes. The act of browsing through newly available data will do far more to clarify requirements than any workshop or session beforehand ever could. Which really underscores the need to deliver part of the functionality as early as possible both to verify as well as to further understanding of requirements. Delivering some working functionality early on also helps to build trust and provide reassurance that the project is (still) on track.

Conclusion
In today’s fast changing business world we need to be ready for change. Today’s knowledge, might be worthless tomorrow. So when we design software or data models, we need to aim for flexibility and sustainability. It is a well established fact that the majority of IT costs go to “maintenance”: accommodating change. Change could be needed because our early prototypes have made clear we didn’t quite get a 100% accurate view on the requirements, or, change might be needed to meet genuinely evolved requirements. Either way, there *will* be change.

For data-driven solutions, the data modeling process is of vital importance. By making it a collaborative and inclusive process, it will safeguard against two kinds of costs. You have “maintenance” costs that are the direct results of missing your requirements sweetspot. Not getting requirements exactly right is permissible, and in fact to be expected, to some extent. Keep all of your stakeholders on board during this process, and they will feel part of the process, and may understand (and forgive) slight errors during this initial phase.

Another maintenance cost you want to keep in check is having clarity around what the solution is supposed to deliver, and how it will work. A data model is a special kind of documentation that represents how stakeholders believe the primary business process is expected to work. Developers embed it in the solution and enforce the relationships it contains in the underlying databases, regardless of whether those are traditional SQL databases, or some of the newer NoSQL solutions. Both use data models in much the same way, but the relationships are enforced very differently.

There may be exceptions, changes and surprises (like in the case of fraud) in the primary business process that don't comply with the data model as stated during the requirements phase. Recognizing and identifying these kinds of exceptions can be illuminating, and potentially valuable.

One of the reasons the authors endorse solutions that allow quick access to both the "raw" source data, as well as the modeled data in the presentation layer is so that you can quickly link any exception values to the underlying source data. This provides a convenient audit trail from raw source data coming in to modeled (transformed) data as they are being shown in reports. This capability can surface relatively innocent data quality problems, or, may enable a second pass at reviewing the requirements. This might have been possible with other approaches, but in particular separating the "raw" data from the representation ("modeled data") that business users expect to see allowed us to untangle business process exceptions from data modeling flaws.

If too many restrictions are enforced in the data model, future changes will result in higher maintenance costs. In this article we've explored three topics that need sufficient consideration upfront, as these will be hard(er) to change in the future. As we laid out in this paper, when you design data driven solutions the topics of Time, Uniqueness, and Role switching between Facts and Dimensions require extra attention. The data modeling choices you make will have consequences, and they need to be understood by all stakeholders involved. Make the data modeling process engaging and fun, quickly demonstrate the results and effects, and everyone will be keen to stay on board!