Business Intelligence Architecture in Support of Data Quality

Tom Breur
XLNT Consulting, the Netherlands

ABSTRACT

Business intelligence (BI) projects that involve substantial data integration have often proven failure prone and difficult to plan. Data quality issues trigger rework, which makes it difficult to accurately schedule deliverables. Two things can bring improvement. Firstly, one should deliver information products in the smallest possible chunks, but without adding prohibitive overhead for breaking up the work in tiny increments. This will increase the frequency and improve timeliness of feedback on suitability of information products and hence make planning and progress more predictable. Secondly, BI teams need to provide better stewardship when they facilitate discussions between departments whose data cannot easily be integrated. Many so-called data quality errors do not stem from *inaccurate* source data, but rather from *incorrect interpretation* of data. This is mostly caused by different interpretation of essentially the same underlying source system facts across departments with misaligned performance objectives. Such problems require prudent stakeholder management and informed negotiations to resolve such differences. In this chapter I suggest an innovation to data warehouse architecture to help accomplish these objectives.

INTRODUCTION

Business intelligence (BI) projects are risky. According to analyst firms like for instance Gartner, or the Standish Group reports, these projects have petrifying failure rates. In large part, these risks are caused by data quality problems that complicate data integration. When data from disparate business silos are confronted for the first time in the data warehouse, this often surfaces heretofore-unknown data quality issues.

BI teams merely 'hold' corporate data (as in: provide stewardship), they do not 'own' them. However, under organizational pressure to complete a project, they feel sometimes 'forced' into a role of pseudo ownership of data (and accompanying data quality errors) while they struggle to integrate fundamentally misaligned source data in the data warehouse.

This problem can either be mitigated, or exacerbated, depending on the architecture chosen for the data warehouse. An optimal architecture for data warehousing should allow interpretation of data to be left to their owners. This is *not* a responsibility of the BI team. They can facilitate discussions about its content, and shed light on source system characteristics, but the BI team should never be tasked to 'own' (as in: commit to long-term choices) interpretation of data quality problems. Interpretation of data, and decisions on how to deal with data quality errors, should be the realm of (senior) management in charge of the incumbent business silos that control source systems.

To (better) deal with quality issues that arise in the course of data integration, in this chapter I will suggest using an architecture that provides auditable and (easily) traceable data transformations. During project development, data modeling choices need to be changeable (at reasonable cost). Currently, the dominant data warehouse design paradigm (Kimball bus architecture) falls short on providing these features.

Data warehouse requirements, often carry considerable ambiguity. To cope with this ambiguity in requirements and resulting changes in the data integration engineering, I recommend a relatively new hyper normalized data warehouse architecture that enables both incremental development, as well as data model reengineering at limited cost.

BACKGROUND

In this chapter I will explain why hyper normalized hub-and-spoke architectures provide better support for data governance and data quality management, and are also better suited to more agile development of BI solutions. We know from experience that BI requirements are often ambiguous, and therefore amenable to change. That's why you need a modeling paradigm that is resilient to change, and that safeguards against a loss of historical information when changes to the data model need to be made over the course of a complex data integration project.

The raison d'être for an enterprise data warehouse (EDW) is to provide integrated historical data, and to be able to do so for a (relatively) long period of time. As such, an EDW is bound to outlive ever changing operational systems, and ever changing reporting needs. In order to be able to do so, the structure of the data warehouse itself needs to be resilient so that it can gracefully absorb these changes that are entirely expected over the course of its lifetime.

In the 90s an 'ideological war' raged between Bill Inmon and Ralph Kimball, two eminent thought leaders from the early years of data warehousing. It is safe to conclude that Kimball emerged as the 'winner' of this debate. Nowadays, the majority of data warehouse architectures are based on Kimball's proposed "bus architecture", a two-tiered data warehouse architecture.

In this chapter I will recommend a revival of three-tiered data warehouse architectures, also called hub-and-spoke topologies. But contrary to Bill Inmon's (initial) suggestion, I recommend *against* using third normal form as the modeling paradigm for the data warehouse. Instead, I suggest using a hyper normalized data-modeling paradigm for the central hub that was specifically designed for data warehousing. For reasons of maintainability and data governance I recommend implementing business rules only *downstream* from the data warehouse (between data warehouse and data marts).

A two-tiered data warehouse architecture transforms source data in one fell swoop straight into data marts that are directly available to end-users. In a three-tiered architecture, an intermediate layer (the "hub" in a hub-and-spoke topology) is built that is (usually) *not* directly accessible to end-users. From there, data marts are sourced. Controversy between these two schools of thought ("Inmon" versus "Kimball") has been whether this intermediate layer is desirable or unnecessary overhead.

I will demonstrate (using artificially simplified modeling challenges to demonstrate the principle) why third normal form modeling in a data warehouse, does *not* handle change very well. This has to do, among others, with the way Inmon (initially) suggested the time dimension should be modeled into the data warehouse, namely by appending a time stamp to the primary key in a third normal form (3NF) model. This causes serious maintenance problems, as practitioners who tried to implement this approach have discovered. Also, changes in the data model can ripple through existing table structures, and cause considerable maintenance overhead (see Figure 1-2).

In this chapter, I will also explain why Kimball's approach to data warehousing (a two-tiered topology, or so-called bus architecture) has important drawbacks, and is intrinsically non-agile. The *primary* reason for this is that the smallest increment of delivery is a star (dimensional data mart), which is considered too

big for frequent and early delivery of value to end-users, as specified by agile principles (Agile Manifesto, 2001). On top of that, the bus architecture forces developers to commit too early which attributes to include, and how to conform dimensions. This defies the principle of delaying decisions to the last responsible moment (Poppendieck & Poppendieck, 2003).

Nowadays, a number of hyper normalized modeling approaches are available that were specifically designed for data warehousing. Examples are Anchor Modeling, Focal Point Modeling, 2G Modeling, or Data Vault. What these approaches have in common, is that core business entities like "Customer", "Sale", "Shipment", etc., are broken out in multiple tables (hence "hyper normalization"). Connecting these associated tables around some natural business key has been labeled Ensemble Modeling (Hultgren, 2012). The manner and degree to which tables are broken out differs between these approaches, but always enables the data warehouse to cope more gracefully with change.

A thriving community of BI experts have emerged that embrace an approach to data warehousing that is often based on some version of the Data Vault modeling paradigm as first described and advocated by Dan Linstedt (Linstedt, 2011). However, the approach I recommend is in no way tied, nor limited to Data Vault modeling. Instead, I suggest using 'a' hyper normalized paradigm that has been specifically designed for data warehousing.

Because of high standardization and self-similarity within the model (Scholten, 2010), Data Vault modeling is particularly well suited for support by automation tools (Bilali, 2013). This has played a decisive role in its adoption. A number of Open Source and commercially sold tools are available for this, and application of such tools has been shown to dramatically expedite development. Model (meta data) driven design shortens development cycle times and also improves code reusability and maintainability.

One of the alternative approaches to Data Vault modeling for three-tiered architectures for data warehousing is Anchor Modeling, as described by Lars Rönnbäck (Rönnbäck, 2012). A suite of code generation tools are also available for Anchor Modeling. Other structured approaches to hyper normalization for data warehousing are likely to emerge in the future in this young and burgeoning field.

It should be noted here that some Industry Logical Models (ILM's) apply forms of hyper normalization, too. Because these ILM's are commercially sold, their structure unfortunately (mostly) isn't part of the public domain, and for reasons of intellectual property therefore cannot be included in academic publications.

Data integration projects face both technical as well as semantic challenges. The technical challenge is finding suitable business keys to match records belonging together. The semantic challenge is interpreting source facts in line with the meaning business stakeholders associated with them. Bridging the semantic gap from (technical) representation of atomic details into a form that represents the business view on reality is (by far) the most challenging task. BI experts refer to this as the "Big T" (short for transformation).

From a project (risk) management perspective, the most daunting challenge is to clarify detailed specifications of *how* this "Big T" transformation should be done. A three-tiered architecture provides the following benefits for this:

- You break up structural transformation for purposes of efficient storage of history (source to data warehouse) separate from the "Big T" (data warehouse to data marts) so that you 'divide and conquer' the workload. This enables you to efficiently advance development in the smallest possible steps.

- You disentangle historizing detailed storage from aggregation and transformation, which allows you to refer data elements directly back to source files as provided to the data warehouse (traceability).
- You disentangle confounding of *interpretation* by this two-step approach because disputes about accuracy of corporate reporting can be viewed in light of raw data elements provided at a specific point in time by an indentified system, compared to report numbers *after* semantic transformation ("Big T") has taken place. This way you can answer whether the 'bad data' in corporate reports were caused by inaccurate data elements that went *into* the data warehouse, or inappropriate transformations that result in erroneous representation going *out* of the BI system.

THREE-TIERED DATA WAREHOUSE ARCHITECTURE

The impact of requirements changes on redesign

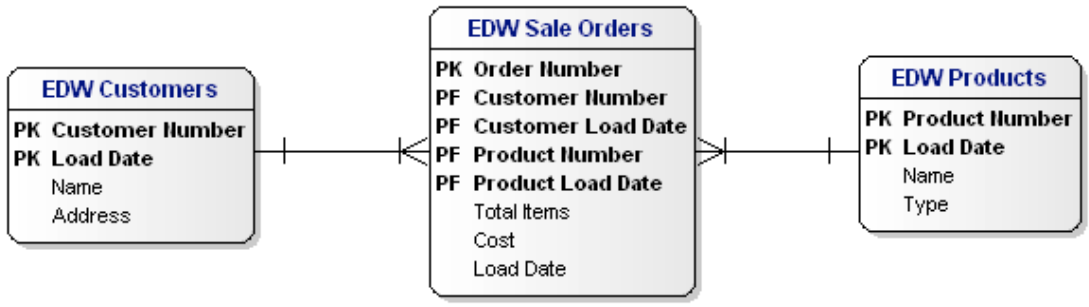
There are a few reasons why hyper normalized approaches to data modeling for the central hub in a three-tiered data warehouse architecture have proven superior. First of all, in agile fashion, BI developers want to embrace change. Experience has shown that a hyper normalized data modeling paradigm can deal better with changing requirements. I will illustrate this with an example.

The fundamental reason why hyper normalized data models absorb change more gracefully is that core business concepts (so-called business keys) are stored separately from their (historical) descriptive attributes, and separate from interconnected relationships (Hultgren, 2012). The effect this has is that model changes stay (and always will remain) *local* so that later updates don't ripple through the model via parent-child key relations.

In the following example, I will demonstrate how this "insulation" from change works. It becomes manifest in the minimal impact when expanding Figure 6 to Figure 7.

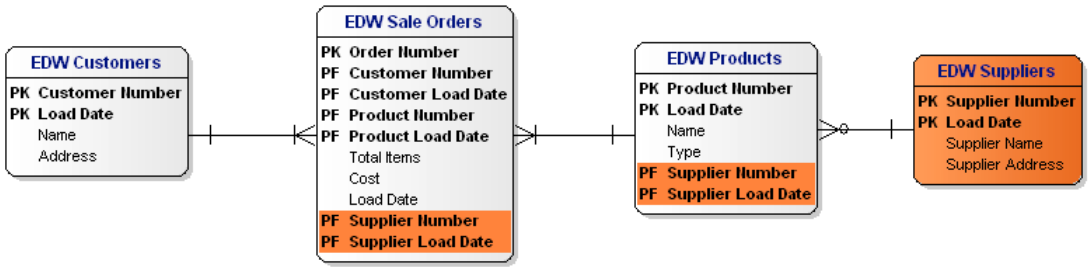
The following figure shows a (simplified) data model for a business selling products to customers, firstly modeled in 3NF (Inmon approach to data warehousing):

Figure 1:



Now let's suggest a 'late' requirement arrives to also report these sales by supplier. This causes a new "Supplier" table to be added, but also note how this triggers an update to the keys in the Product and Sales table.

Figure 2:



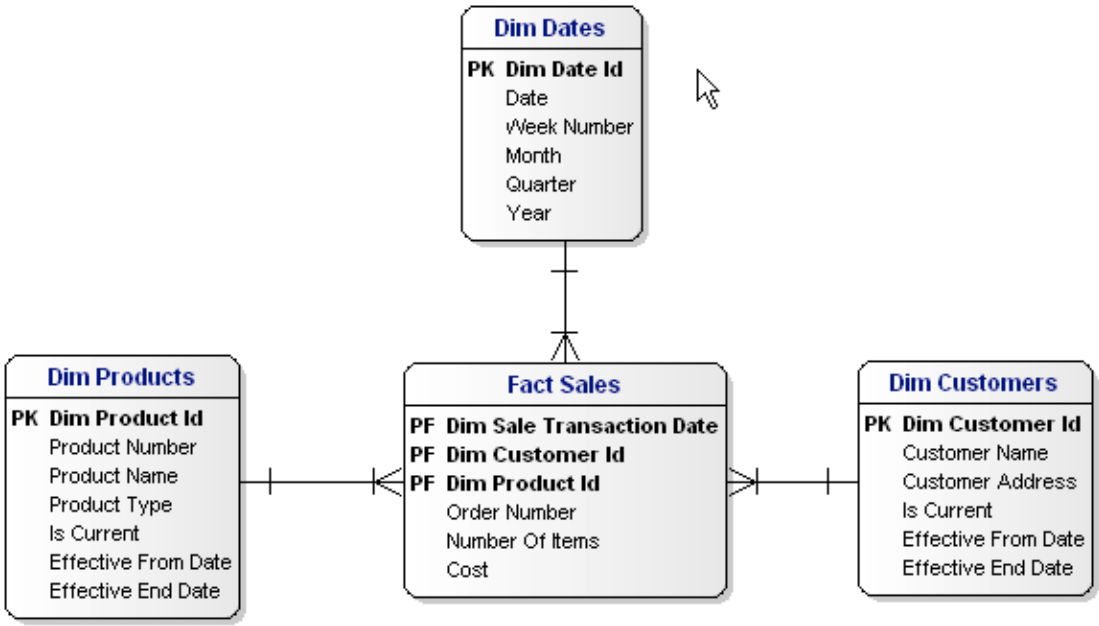
Because in 3NF business keys and relations are stored *together* in one table, changes therefore ripple through the model when parent-child relations propagate updates. This makes it difficult and expensive to contain the risks of (unforeseen) changes in requirements when these impact the structure of the data model. This can easily happen if the scope of requirements change.

3NF modeling is also undesirable for another reason. It makes it hard to determine data model boundaries. That is why it doesn't support incremental development very well, which agile developers value. They want to be able to start very small and efficiently grow in tiny increments, without adding significant overhead because of these 'little baby expansions'.

Because 3NF modeling doesn't facilitate incremental development very well, scoping out small iterations is difficult *and* adds overhead. Data warehouse projects that attempt to 'boil the ocean' take too long to deliver value, and fail to reassure stakeholders about progress because they cannot demonstrate tangible results early on. Big releases also bombard developers with change requests during the testing phase. This is less than prudent given known project risks in BI.

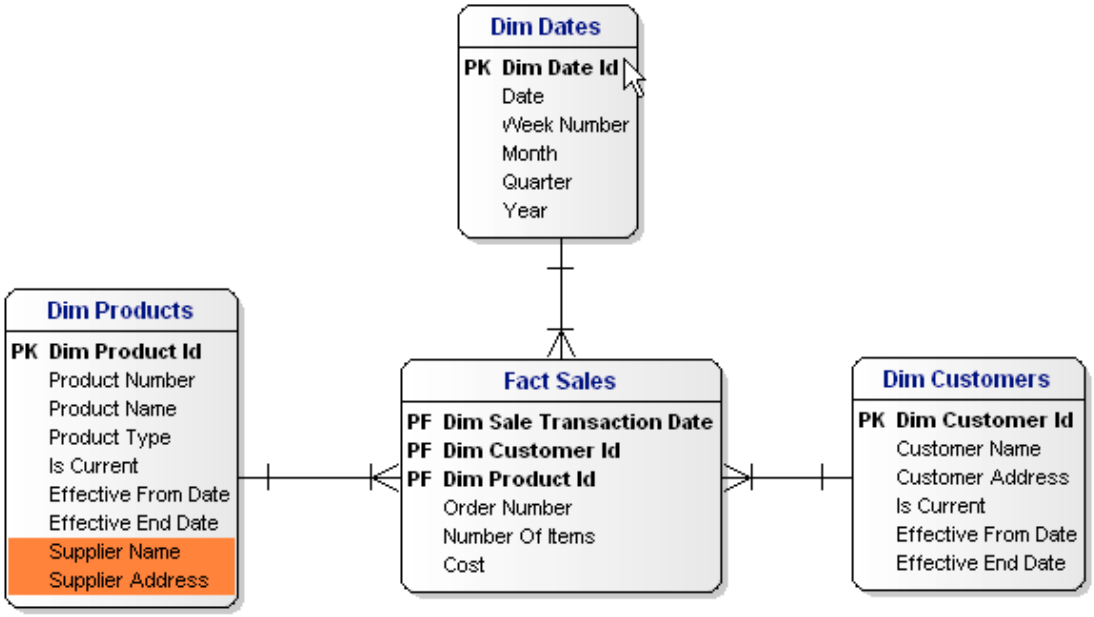
The previous reengineering problem would look slightly different in a dimensional model (Kimball approach to data warehousing):

Figure 3:



If the supplier data have a one-to-one relation to products (each product is procured from only one supplier), they could get added to this data model by 'simply' appending suppliers to the product dimension. This results in the following diagram:

Figure 4:



This would constitute only a small change to the data warehouse. All this requires is that the ETL for the product dimension needs to be updated, and (besides possible impact on downstream reporting) the data warehouse could continue to run very much just like before. When business stakeholders assure this one-to-one relation between product and supplier always holds, many BI teams would happily implement this solution.

Drawbacks of implementing business rules *in* the data warehouse

The design in Figure 4 works fine if, and only if the one-to-one relation between product and supplier is maintained for *all* products. This might well be a viable solution in an operational system, because in most operational systems there is not too much concern for historical changes. You are typically concerned with ensuring the right supplier gets marked for each product, and the fact that a product might have been procured from different suppliers over time is (usually) less of a concern (for operational purposes).

But there is another, more subtle problem with the solution in Figure 4, one that is of major concern from a data governance perspective. The relation between product and supplier is now 'hard wired' into the structure of the data warehouse. What this effectively does is that it *enforces* a structure on data arriving into the data warehouse. But this is not the role of the BI team, to enforce what is or is not allowed to happen in source systems.

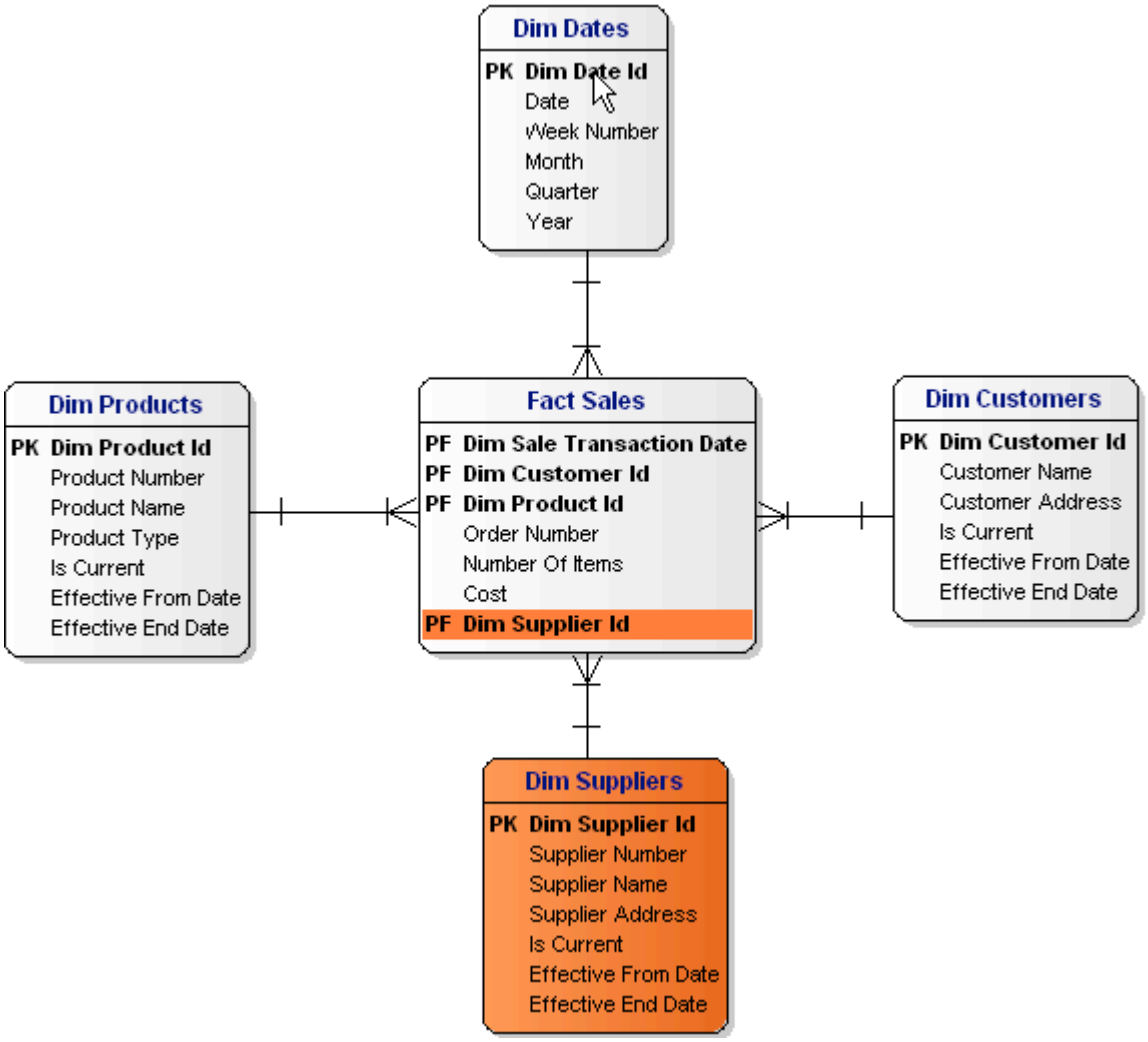
BI professionals should report what has happened, not impose rules on what is 'allowed' to happen in real life. The former is the role of BI professionals; the latter is the responsibility of business management. In fact, it is important that a BI team retains the capability to report on what *has* happened (including exceptions), even though such events *should* not happen. The BI team should always strive to remain data agnostic, or else they would risk loosing their independence.

When business rules are imposed on data arriving into the data warehouse, the affected data are not loaded into the data warehouse, but instead are dropped in exception reports during loading of the data warehouse. Auditability in the data warehouse is lost, and the independence of the BI team is compromised. There are also other concerns with implementing business rules upstream from the data warehouse that I will address in the section "Consequences of business rules upstream from a data warehouse."

The impact of requirements changes on redesign (continued)

When products can be procured from several suppliers, *and* the business would like to make comparisons across suppliers, then the solution in Figure 4 does not work. What is needed in a dimensional model is adding a supplier dimension. Note that in this scenario, the impact would be considerable. Updating the "Product" table here, *also* affects the grain in the central fact table. Before this change there was only one record per product in the "Sales" fact table. After adding the "Supplier" dimension, there will be separate records for every supplier.

Figure 5:



This problem is known, and has been documented (Mundy, 2012). It results in redoing the initial load of the data warehouse. For large data volumes, this becomes a (very) resource intensive task. Because data volumes grow over time, the cost of redoing this initial load also grows over time.

Another reason why redoing the initial load is so daunting, is because changes in the structure of the source files, or, changes in the way their content should be interpreted, require going through multiple versions of extract, transform, load (ETL) packages as used over time. For sequential sets of source input files, sequential versions of ETL need to be called, in accordance with corresponding periods of history. This becomes a serious maintenance challenge, in particular as more time goes by.

End-users don't always have their requirements clear at the outset of a project. Agile methods suggest we should make it as easy (and inexpensive) as possible for end-users to *change* their requirements. We want to *embrace* change, rather than resist it. Underlying business needs often only emerge in the course of *using* BI solutions, which makes it fundamentally impossible to provide stable requirements upfront. In this context, Corr (Corr & Stagnito, 2011) speaks of BI requirements being *accretive*, gradually building up over time like layers of sediment. BI architecture should support this design agility goal, and should not make it prohibitively expensive for end-users to change their mind later on.

The case for a three-tiered data warehouse architecture

For the reasons I explained above, late changes to a data model that was designed using the Kimball bus architecture tend to be cumbersome and expensive. They require keeping a full history of all loaded tables in a persistent staging area, all versions of ETL ever used, and careful engineering while redoing the initial load.

One of the Lean software engineering principles is delaying decisions until the last responsible moment (Poppendieck & Poppendieck, 2003). This principle is used in conjunction with concurrent development to enhance agility. Decisions that haven't been made, yet, never need to be reverted, and therefore carry zero reengineering cost.

A fundamental problem with two-tiered architectures is that (all) dimensions need to be conformed *before* you can make data available to end-users. You need to decide which attributes to include upfront, because adding attributes afterwards requires that you either:

- Set this value to missing for the time slice prior to including it in the data warehouse. This is a misrepresentation when in actual fact its value is known;
- Set this attribute to the *current* value (treat it as a Type I Slowly Changing Dimension, SCD), which most likely is wrong because if this value never changed there would probably be no reason to include it;
- Redo the initial load, this time including the new attribute.

None of these options look pretty, and they were all caused by a late arriving requirement for this new attribute. If you want to avoid this risk, you will need to decide *early* which attributes to include. By the same token, you will need to decide *early* how to conform dimensions in a two-tiered architecture.

I recommend three-tiered hyper normalized data warehouse architectures for the following reasons:

- Such architectures facilitate incremental delivery in (very) small steps, without adding excessive overhead;

- Such architectures enable concurrent development, and allow developers to delay (risky) decisions like how to conform dimensions, and which attributes to include, to the last responsible moment;
- Design commitments can be reverted, even late in the development process, at limited cost;
- Reengineering in the data model is guaranteed to remain localized, thus containing the cost of late changes;
- A three-tiered architecture (with business rules enforced *downstream* from the data warehouse) provides superior support to data quality enforcement and data governance.

This last "business rules downstream" argument merits some elaboration. Business rules are *always* needed to transform data in source systems into some representation that is meaningful to end-users. The source system will often be a 3NF structure, the target will usually be a dimensional structure as this facilitates access for end-users. The problem with implementing business rules *upstream* from the data warehouse is manifold: maintainability, requirements gathering, but the issue we are most concerned about with regards to data governance is (immediate) traceability and auditability.

For reasons of data governance, it is vitally important that for all data elements in the data warehouse, a record source and date/time stamp of arrival be appended. This will provide reference to a specific source file that was provided by a particular system with an immediately available date and time when these data were received by the data warehouse team.

This poses somewhat of a conundrum: traceability is a requirement that is imposed by the data warehouse team. It is not (usually) an *explicit* requirement from the business sponsors of a data warehouse project. This need ("requirement") follows from an *implicit* business requirement that all data in the data warehouse be trusted (reliable). This accountability of each and every data element will eventually cause successful data warehouses to become a system of record for the state of affairs throughout the business, because most source systems only keep a partial (and non-auditable) trace of history. When designed in the way I suggest, the data warehouse will contain the entire corporation's history for all data included.

The immediately available data lineage of each and every data element to identified systems, and specifically marked source files as supplied on a given date and time, allows the BI team to take their steward role very serious, yet at the same time avoid becoming (being seen as) 'owners' of corporate data.

Consequences of business rules upstream from a data warehouse

When testing data quality in the warehouse and in data marts, there is a confounding between accuracy of source data provided, and the impact of business rules when transforming data for 'consumption' by end-users to its desired shape and form. This has a negative impact on data governance, which is why I recommend embedding business rules exclusively *downstream* of the data warehouse to facilitate data quality work, and hence better support data governance.

Immediate backward data lineage from the output of business intelligence reports back to data files as provided by source system owners enables a BI team to stay "semantics agnostic" with regards to data interpretation as required for integration in the data warehouse. Such an architecture allows the team to focus on and limit themselves to their data stewardship role. When someone expresses his doubts about the accuracy of a corporate report, it is highly beneficial and enlightening to immediately be able to relate that report to sources files as provided by an identified supplying system on a particular date and time.

Because of elaborate sequential transformations in the ETL layer, which passes form source systems straight into data marts, Kimball style architectures *include* interpretation as part of data storage. There is no evident lineage from a report to a 'raw' number that was received from some source because of the

business rules embedded in ETL. In a bus architecture, business rules need to be implemented *upstream* from the data warehouse.

A disadvantage of Bill Inmon's original data warehouse architecture (although it uses three layers), is that business rules are embedded upstream *as well as* downstream from the data warehouse. In this architecture, the transformation goes through two stages, a "semantic" one from source to data warehouse hub, and a "selection/aggregation" transformation from the data warehouse to data mart. This (severely) hampers auditability and traceability of data because the upstream business rules have impacted what data made it to the hub (and how).

In my proposed architecture, there should always be direct and straightforward traceability of data to their sources. The type of hyper normalized data model I recommend provides only structural change to the data to efficiently store a historized version of the data. The content is not semantically transformed or modified on the way from the source system to the data warehouse hub. *No* business rules are embedded upstream from the data warehouse. So all data are loaded, all the time. The good, the bad, and the ugly. Data are integrated around business keys that are already commonly used throughout the business to refer to relevant corporate entities (e.g.: "Product", "Sale", "Shipment", etc.).

Origins of data quality problems and business alignment

A fundamental reason why three-tiered architectures like I propose provide better support for data quality objectives, has to do with the overarching corporate objective to support consistent, company wide decision-making. BI should strictly limit itself to fulfilling a stewardship role. To make a "single version of the truth" possible, BI teams need to conform dimensions. This holds regardless of whether they use a two-tiered *or* three-tiered architecture for data warehousing. Conforming dimensions is one of the areas where many purported 'data quality' problems arise.

Data quality problems stem largely from one of two sources: either inaccurate data entry, or semantically incorrect *interpretation* of data that are essentially correct from a technical point of view for a particular source system. I argue the latter category constitutes the majority of purported 'data quality' errors, that are in reality semantic interpretation errors.

The single most important driver for these data interpretation discrepancies is misaligned conforming of dimensions. How to conform dimensions is *largely* a political corporate discussion. It hinges on two (or worse: even more) departments agreeing to a shared 'view' on the facts as they have been surfaced from source systems by the BI team.

Note that *not* conforming dimensions doesn't provide a solution, either. In that case, two seemingly similar constructs, e.g.: "Customer" as defined by Marketing, versus "Customer" as identified by Finance will each have their own dimension. However, because of (slight) differences in definition, (typically small) differences in counts will result when you query the same facts using these two different "Customer" dimensions. Antithetical to our desired "single version of the truth."

The way that dimensions should be conformed can rarely be decided at the outset of a data warehouse project. If conforming dimensions were easy, then you wouldn't have a challenge to integrate data. This is for the exact same reasons as why there is no single version of the truth, either. Converging on a *shared* interpretation of a dimension *prior* to building a data mart (with these conformed dimensions) is hard, because when no data are available to end-users, they cannot carry an *informed* discussion on the impacts of settling on various options for how to define this dimension (e.g.: "Customer").

This challenge surfaces a fundamental flaw in the bus (two-tiered) architecture: the BI team needs to facilitate a discussion between two or more stakeholders of the data, on *how* exactly to conform dimensions, i.e., how to define the meaningful entities as described in dimensions for an enterprise wide data warehouse. Precisely because there *is* no commonly agreed upon enterprise wide definition of these entities, data integration is such hard work. For the same reason, master data management (MDM) projects struggle to come to completion. The BI team should *facilitate* these discussions, but carefully avoid not to (prematurely) take ownership of the eventual decision.

However, in the early stages of a data warehouse project (when using a bus architecture), none of the business end-users have access to data because the BI team hasn't finished building their solution, yet. Therefore, end-users cannot run reports using one or the other definition for a dimension to see what impact these choices will have on the numbers as they appear in corporate reporting.

Because data aren't available to end-users, yet, the BI team falls short in facilitating an informed discussion between stakeholders throughout the business. If they chose a two-tiered architecture, they 'have' to (prematurely) take ownership themselves for how they choose to conform dimensions. Either they build and deploy a pilot solution, or they take hypothetical guesses on what the quantitative impact will be of one choice over another. Both of these outcomes are both highly undesirable, and largely the result of being 'forced' into early design decisions because of the nature of data propagation in a two-tiered data warehouse architecture.

This ownership for *how* to conform dimensions, gets the BI team caught smack in the middle of corporate cross-fire: no matter how you define an entity in a conformed dimension, 'the numbers' (relative to management performance objectives) will *always* come out better for one manager (or department) than for another. Hopefully you were either very smart or lucky, and happened to choose a definition that favors the more influential stakeholder. But if you consistently need luck to stay out of trouble, it becomes only a matter of time before 'Murphy' strikes.

What tends to happen in most data warehouse projects using the 'traditional' Kimball (bus) architecture, is that the BI team commits to a certain view on how to conform dimensions, but this commitment needs to be made too early. There appears no way out of this conundrum without making 'a' decision. The BI team otherwise cannot proceed with development of their data mart.

Making changes to a Kimball data warehouse becomes prohibitively more expensive as time goes by because the volume keeps growing every day. Furthermore, complexity keeps growing as changes to source file structures (e.g.: additional attributes added) necessitate updating ETL code. Therefore, the BI team finds itself 'locked into' a design, and looses flexibility. This situation becomes even more problematic when data quality issues don't surface until much later.

The dynamic at play here is that delays in releasing data for corporate 'consumption' (or acceptance testing) delays feedback on the appropriateness of the delivered solution. This feedback from end-users is essential to keep development on track. By making increments of delivery as small as possible (design agility), you effectively shorten these feedback cycles. This ensures timely changes, and minimizes the amount of rework in case a transformation *does* need to be corrected. Resolving issues quickly and progressing piecemeal is significantly more efficient.

Note that in a three-tiered architecture you might *still* need to rebuild a data mart using new (adjusted) transformation rules and/or using a new definition for our conformed dimension. However, you do *not* need to redo the initial load because all of those data are (still) available in the central hub.

Note that for these same reasons, many BI teams resort to virtual data marts. Deploying data marts as Views avoids physically copying data, which makes the release of alternative renditions of these data marts so much quicker and more agile. All that is involved is changing a few lines of code and releasing this update.

This approach using virtual data marts usually does not work for two-tiered architectures. Besides performance challenges when you relay data directly from source systems straight into a data mart, most source systems don't keep a comprehensive history of values required for time variant reporting.

Conforming dimensions is *largely* a political corporate discussion. These discussions become 'risky' when (individual) performance objectives of incumbent managers are at stake. He who finds himself with a less favorable number in corporate reports (relative to his individual target) will quite 'naturally' resist this new view on how to conform the dimension.

We, as BI professionals, don't own the data. We merely hold them for the business. As stewards it is our responsibility to surface discrepancies in definitions, and quantify where and how the business is loosing money as a result of this confusion (i.e.: these data quality issues). When an acquisition department has captured details for customers that Marketing doesn't 'recognize' (because they use a slightly different definition of "Customer"), those prospects are effectively lost for further cross- and up-sell opportunities. Acquisition efforts and money are wasted. Those cracks in the corporate value chain are invariably quite interesting to senior management, in particular when you are able to quantify the magnitude of these losses.

Data quality problems don't often originate from truly 'technical' causes. I argue that mostly, purported data quality problems are fundamentally the result of poor (internal) alignment of corporate objectives. These misaligned objectives place corporate stakeholders at odds because for some of them, changing the definition for a business entity (e.g.: "Customer", "Sale", "Shipment", etc.) causes their performance to look worse (at least for some).

But sometimes, data quality problems *are* the result of sloppy data entry. Still, this can usually be traced back to a misalignment of corporate objectives. The classic, albeit hackneyed example is rewarding people for speed of data entry, rather than for its accuracy. The back office manager (problem owner) is expected to process as much work as possible, with the least amount of resources. He is rewarded for speed. Downstream information users (problem holders) bear the brunt of hasty work, and failure to install ("time consuming") inspect-and-adapt cycles to prevent data entry errors. They need accuracy. This, again, points to misaligned performance objectives for the problem owner (back office manager) who is expected to work 'efficiently' (i.e.: minimize resource usage), and downstream information consumers (problem holders) who require accurate data to be able to do their job properly.

The grand perspective on this dynamic goes further. The majority of data quality issues point to cracks in the corporate value chain. In this chapter I argue that information loss (either 'errors' or inappropriate reporting) is mostly caused by a failure to conform dimensions (politically: align corporate objectives). In that way, BI team members take on the burden of misalignment between cross-departmental objectives when they feel 'forced' to choose sides in a corporate discussion on how to interpret facts in the source systems. This is unnecessary if that results from their choice of warehouse architecture, i.e., the Kimball bus architecture. It doesn't have to be that way.

How to avoid these problems?

If you store all the (source) facts in your data warehouse, you can keep them available in more or less original state. By this I mean *efficiently* with regards to historizing data (capturing all changes in source

data, with minimal redundancy) and also *flexible* with regards to anticipated changes and inherent ambiguity of requirements. The data warehouse hub should contain at the most only basic (non-semantic) transformations so that all data elements remain traceable to the supplying source system and specific file provided. Data remain auditable when no business rules have been applied. Then there is a way out of this conundrum. This requires you move away from 20th century data warehouse architectures that have required BI practitioners to conform dimensions (too) *early* in the project.

The practice of combining data integration and transformation in one fell swoop (e.g.: a two-tiered architecture) puts ETL developers at a disadvantage, because requirements are often ambiguous (when end-users cannot 'see' the proposed solution, yet), and amenable to change. If the cost of change over time then further becomes prohibitive (because redoing the initial load becomes ever more cumbersome), you take irresponsible risks by committing too early which attributes to include and how to conform dimensions. Instead, you would like to achieve the opposite; you want to mitigate those risks. We want to keep the costs of change low to enable business stakeholders to change their mind at limited cost, throughout the course of data warehouse development.

Delaying decisions to the last responsible moment

In agile software development practices we like the idea of "deciding at the last responsible moment", in conjunction with concurrent development practices (Poppendieck & Poppendieck, 2003). That way, we can show end-users bits of functionality (very) early on. This gives us credit with our sponsors, and business stakeholders will gain confidence in our progress. But at least as important: the sooner reliable feedback about data accuracy is passed back to the development team, the more efficient and effective they will be able to make necessary (small) changes and continue with their work.

Imagine you could run your BI projects like this. You inform all departments involved what the consequences are (counts, frequencies, amounts, etc.) of choosing various definitions to conform data across systems and departments. That way, you empower all stakeholders to have an informed discussion about what 'view' on the facts would best support *their* particular (end-to-end) value stream.

All business stakeholders have good reasons to cling to their departmental specific view on the source system facts. When you apply this approach to managing BI project risk, stakeholders are well-placed to explain why the numbers for (global) corporate reporting don't exactly line up with the numbers for any specific department. This doesn't necessarily point to any (technical) data quality errors, there may well be *business process specific* reasons for these discrepancies. Not only can these reasons be alluded to, by utilizing a three-tiered architecture in the way I have outlined (with business rules embedded *downstream* from the data warehouse), you can *also* report on the magnitude of the differences and shed light on the business case for continuing department-specific interpretation of source system facts.

Conversely, one can also imagine that when you attempt to conform dimensions *before* 'the business' has agreed on common dimensions, this leads to dispute that can only be solved elegantly by providing counts under different scenarios, using (slightly) different definitions for the conformed dimensions. Uninformed arguments over 'the right' way to conform dimensions can have a detrimental effect on the progress of a data warehouse project. Potentially, the BI team might bear the brunt of their frustration when business stakeholders are confronted with 'erroneous' reports.

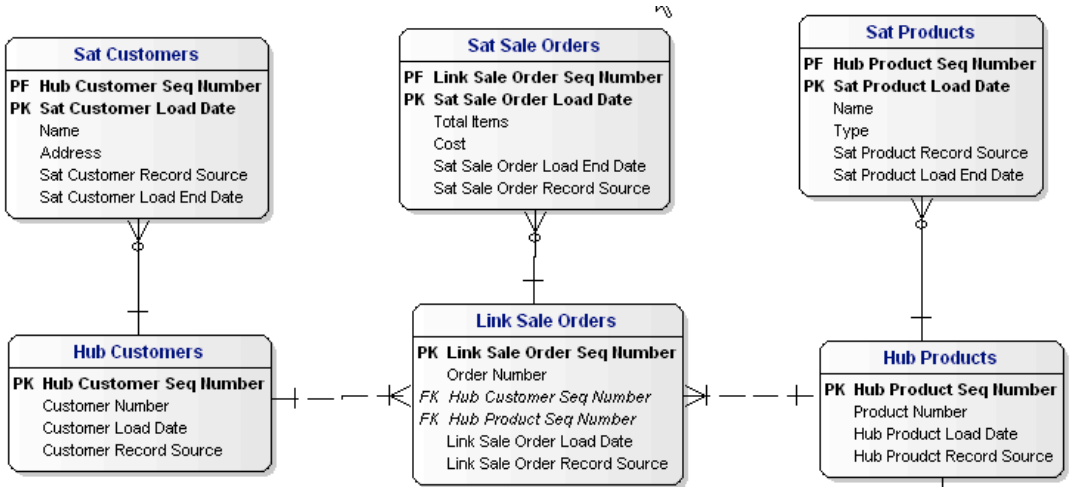
When BI teams inadvertently become 'owners' of corporate data and their interpretation, this creates a problem in business alignment. Every time you become problem holder (the person experiencing pain) without adequate resources to solve the problem at its root (the problem owner), you risk getting caught in the (corporate) line of fire. Fundamentally, this problem was caused because the BI team chose to decide how to align data from different data providers (business silos), without access to the resources to resolve

the underlying discrepancies. BI teams can prevent this straddle by moving from a two-tiered architecture to a three-tiered architecture with traceable and auditable data contents in the central hub.

A hyper normalized three-tiered architecture

As I have outlined before, to make a data model resilient to change, you need to break out business keys in separate tables from the relations, and separate from descriptive attributes (and history). I will now show how such an approach might look if you apply Data Vault modeling. To begin with, I show the same initial example again with only "Customers" and "Products", but (first) without "Supplier."

Figure 6:

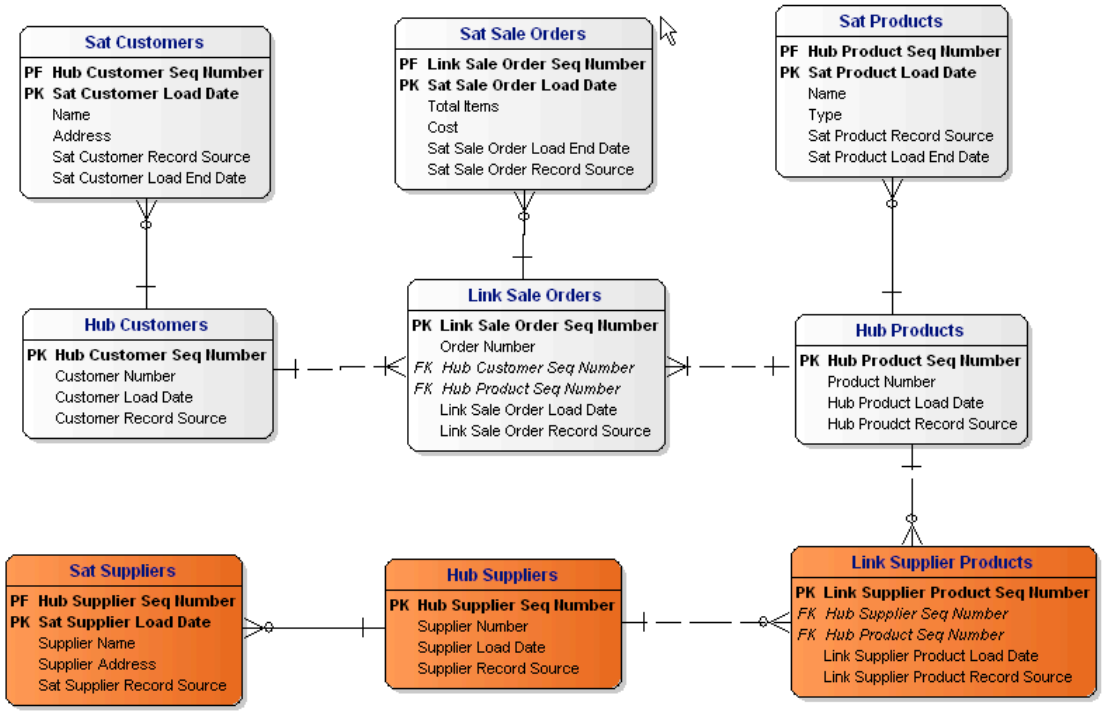


The first thing that is apparent from Figure 6 is that you now need more tables to describe the (exact) same data structure we saw in Figure 1 (3NF model) and Figure 3 (dimensional model). In this particular example, the business event "Sale" has been modeled as a (many-to-many) relation ("Link") between "Customer" and "Product", with descriptive attributes for "Customer", "Sale Order" and "Product" represented in their respective tables "Sat Customers", "Sat Sale Orders", and "Sat Products."

What is also apparent is that query paths through this model will contain more joins, and therefore will have slower response times. To some extent this degradation in performance will be attenuated by query optimizers (available in almost all relational database management systems) that execute selective joins. If a join that is *logically* present in a query path does not affect the selected set, it will be dropped from the *executed* query, hence improving (optimizing) performance.

Now let us see what happens in this hyper normalized model when a new business requirement surfaces to include "Supplier" data into the model:

Figure 7:



A few conclusions become apparent when you compare Figure 7 to Figure 6:

- The existing data structure ("Hub Product") was not affected in any way by appending the "Supplier" data;
- The many-to-many relationship between "Hub Products" and "Hub Suppliers" does not enforce any business rule on the relationship between these two entities.

Because the "Link Supplier Products" represents a many-to-many relation between "Products" and "Suppliers", both a one-to-one, and one-to-many relation can be captured in this data model. Bear in mind that even *if* the business reassures us that there is a one-to-one relationship (similar to Figure 4) between "Product" and "Supplier" (every product is only procured from one supplier), there is still no enforcement of this business rule in the data model. If such a relationship were to hold, this will become apparent in the data. The data model does not enforce this business rule on incoming data. So when this relationship changes over time, and the business starts to procure products from multiple suppliers, this has zero impact on the data model (it requires no change).

What is not (immediately) apparent from Figure 6 and Figure 7 is that when later additional (descriptive) attributes need to be included in the data model, this can be done in one of two ways:

- Attributes can be appended to an existing Sat table;
- An additional Sat table can be modeled to form a one-to-many relationship with the corresponding Hub or Link table.

The former would require some reengineering of the existing Sat ETL code, and the history of this new attribute would be missing for dates prior to being added to the Sat (or could be set to the current value). Including history in the existing Sat table would require the same type of complex reengineering we saw in a dimensional model when attributes are added to the data model later on. The latter (adding a new, additional Sat) can be done with *no* reengineering, and (all) the history that is still available for this attribute can be easily loaded for this attribute, using the default ETL loading procedure for this table structure.

This design flexibility in case a need arises to add new descriptive attributes to the data model gives data warehouse architects elegant options to quickly (and inexpensively) respond to changing requirements.

FUTURE RESEARCH DIRECTIONS

Data warehousing is (still) a fairly young discipline. Design patterns for how to effectively build maintainable solutions are only beginning to see the light. The inherent self-similarity in hyper normalized data models lends itself particularly well to automation. It has proven possible to generate code, which has several advantages. Generating code can be done by using templates, or by using data warehouse automation tools. This has advantages both in terms of development speed and maintainability of code.

There is impressive anecdotal evidence for efficiency improvement as a result of using data warehouse automation tools. An improvement factor of 5-10 times has been reported. It would be good if these claims, and the preconditions required to achieve such benefits could be substantiated in carefully controlled scientific experiments.

Ensemble modeling is a new data modeling paradigm particularly well suited for data warehousing that has evolved into a number of 'flavors' like Data Vault, Focal Point Modeling, or Anchor Modeling, for instance. As of yet, we lack sound empirical evidence on how they compare when it comes to handling changing requirements. These approaches have slightly different requirement gathering methods, which might have an effect on the 'likelihood' that late changes might trigger updates to the data model. Some approaches are more restrictive in terms of prerequisite information analysis, and will therefore mean that developers will spend more time in this initial requirements gathering phase. Is that time well spent? Or should progress rather be pursued by commencing sooner on the basis of lightweight requirements? The jury is still out, and scientific research could shed light on these questions.

All of these approaches employ some form of hyper normalization. Therefore, you may expect to find more tables (it is estimated a factor 2-3, and more for Anchor Modeling) to model the same business process. These additional tables and joins are largely mitigated by the intrinsic self-similarity in these data models. There are only a (very) limited number of loading and querying patterns, which is why templates or automation can easily support these approaches. This is where spectacular efficiency gains originate. Again, how different modeling approaches compare in that respect remains to be explored. How performance is affected in practice is relevant, but again, no empirically sound research seems available to date.

CONCLUSION

Data warehousing is a relatively new and dynamic field. Ralph Kimball initially emerged as the 'winner' from the "big debate" in the 90s over proposed data warehouse architecture. The two-tiered approach to architecture that he proposed has almost become the industry standard. Most data warehouses nowadays are built using the approach he laid out in his "Toolkit" book series.

Although I acknowledge and support that dimensional modeling will usually be the preferred way to present data to end-users, I have put my case forward in this chapter why three-tiered data warehouse architectures are superior in support of data quality and data governance.

Besides recommending a three-tiered architecture, I also suggest you avoid implementing business rules upstream from the data warehouse. This hampers auditability and traceability of data, which is clearly needed to support data stewardship when questions arise over the correctness of reporting.

There are two essential reasons for my endorsement of three-tiered architectures. Firstly, the three-tiered architecture enables incremental agile development in tiny steps without adding prohibitive overhead. What this accomplishes is that feedback loops to indicate accuracy and usefulness of delivery are dramatically shortened. This helps to keep risky BI projects on track, or when timelines are in jeopardy, at least you are notified early. Secondly, the three-tiered architecture with business rules downstream provides superior support to data stewardship work in BI teams.

The confounding between accuracy of input data (as provided by source systems) and appropriateness of transformations ("Big T" ETL) can be resolved by providing auditable traceability to data elements in the data warehouse hub. These data are marked with their supplying source system and date/time of origin. When the accuracy of reports gets questioned by business stakeholders, you can immediately refer to the data that went into the data warehouse, and compare that with what had been expected. If the input data were correct, you may then compare transformations imposed on source data to resolve ambiguity in requirements or detect errors in transformation logic.

Hyper normalized data models cope (much) better with change. Over the lifetime of a data warehouse change is to be expected. Systems change, data structures change, so your choice of data model should cater to that. My simplified example has demonstrated superior resilience to change. Those effects become (much) bigger in real life applications.

In the architecture I propose, an integral traceable history of all source data elements is persistently captured in a central data warehouse hub. Downstream data marts may contain physical extracts, or, can be delivered virtually (as Views). The advantage of virtual data marts is that updates to its transformation logic can be deployed (much) more rapidly. This enables (quickly) running reports using different definitions of conformed dimensions to assess what the impact will be, and how they compare.

Business intelligence professionals have been pursuing a so-called "single version of the truth." This requires conforming dimensions. In order to arrive at corporate-wide agreement on how to conform a dimension, you need to persuade *some* stakeholders to adjust the way they have been defining business constructs (e.g.): "Customer", "Sale", "Shipment", etc. The most elegant way to get there is through a democratic process where all parties have information about the implications and consequences of making such changes. Having auditable and traceable data readily available in your data warehouse provides an excellent starting point for facilitating such discussions *before* you make irreversible design commitments.

REFERENCES

Beck, K. et al (2001) Agile Manifesto. Retrieved 20 December 2012 from http://www.agilemanifesto.org

Bilali, L. (2013) Data Warehouse Generation Algorithm Explained. Retrieved 17 February 2013 from http://www.datawarehouseautomation.org

Corr, L. & Stagnito, J. (2011) *Agile Data Warehouse Design: Collaborative Dimensional Modeling, from Whiteboard to Star Schema*. Leeds, UK, DecisionOne Press.

Hultgren, H. P. (2012) *Modeling the Agile Data Warehouse with Data Vault*. Denver, CO, New Hamilton Press.

Linstedt, D. (2011) *Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault*. Albany, NY, CreateSpace Independent Publishing Platform.

Mundy, J. (2012) Design tip #149 Facing the Re-Keying Crisis. Retrieved February 22, 2013 from http://www.kimballgroup.com/2012/10/02/design-tip-149-facing-the-re-keying-crisis/

Poppendieck, M. & Poppendieck, T. (2003) Lean Software Development: An Agile Toolkit. Boston, MA, Addison-Wesley

Rönnbäck (2012) Retrieved 20 December 2012 from http://www.anchormodeling.com

Scholten, H. (2010) Musings on the Data Vault. Retrieved 20 December 2012 from http://www.bi-team.com