What value do we get from having an intermediate hyper-normalised layer?

Let me start by stating that a Data Warehouse is a necessary evil at the best of times. In the ideal world, there would be no need for it, as optimal governance and near real-time multidirectional data harmonisation would have created an environment where it is easy to retrieve information without any ambiguity across systems (including its history of changes). Ideally, we would not have a Data Warehouse at all, but as an industry we have a long way to go before this can become a reality.

The other day I was discussing Data Warehouse architecture and collaboration topics during a phone conference and the following question was asked: 'would it be worth to purchase any of the books on Data Vault (or similar hybrid modelling approaches)?'.

Apparently, some of the book reviews were fairly negative by way of dismissing the Data Vault methodology as being past its prime. The question that came up during the call is one that is frequently recurring: would it still be worth investing time and energy in Data Vault or would time better be spend on working with, say, Python or Scala to directly develop data solutions given the current maturity of the industry?

Isn't it just a hassle having to manage an additional Data Warehouse layer when you can directly code against the raw data?

Data Warehouse Virtualisation also appears to contribute to the idea that you don't need a Data Warehouse (which I find to be a bit of a paradox). When you have 'modularised' all data integration components and can dynamically distribute infrastructure resources, rebuilding the Data Warehouse on the fly, does it still make sense to develop a normalised Data Warehouse layer with a technique such as a Data Vault? Can't we just skip it?

I recently touched on this topic while addressing common questions in this post, but it's worth exploring further. This is because there still seems to be is a 'mindset gap' between the 'new' world of 'big data' technologies and the more traditional world of developing Data Warehouse solutions on RDBMS technologies. And, the 'new' world often states it replaces the need for the 'traditional' one.

The opinion is sometimes that these worlds aren't compatible, but my view is that there is no reason why this would be the case. They contain a mutually complementing set of skills and can greatly benefit from a unified approach.

To cut a long story short: my answer to the posed question would be that, no, Data Vault is not obsolete and will not be for a while as it has a role to play in this unified approach. This is also the case for similar hybrid modelling techniques. However, our thinking has evolved quite significantly over the last few years to the point that the first generation of books may not be accurately capturing the intent anymore.

Moreover, the core mechanics of data handling are increasingly pushed down into the background as a commodity – which is a decidedly different focus than physical data model design. Data Warehouse Automation software and frameworks such as BimlFlex (from Varigence, the creators of Biml), WhereScape and BIReady are continuously working towards this, as are my own (Github / collaboration) efforts on 'Virtual Data Warehousing'.

In short: Data Vault seems destined to take a place in the technical *delivery* of data solutions, but less as a technique to *design* the data models. Data Vault has proven itself as a capable technique to support incremental, business focused and tightly scoped information delivery. It provides a strong foundation to base your designs on, but physical design itself should never be the focal point of data solutions. Information modelling and management should be.

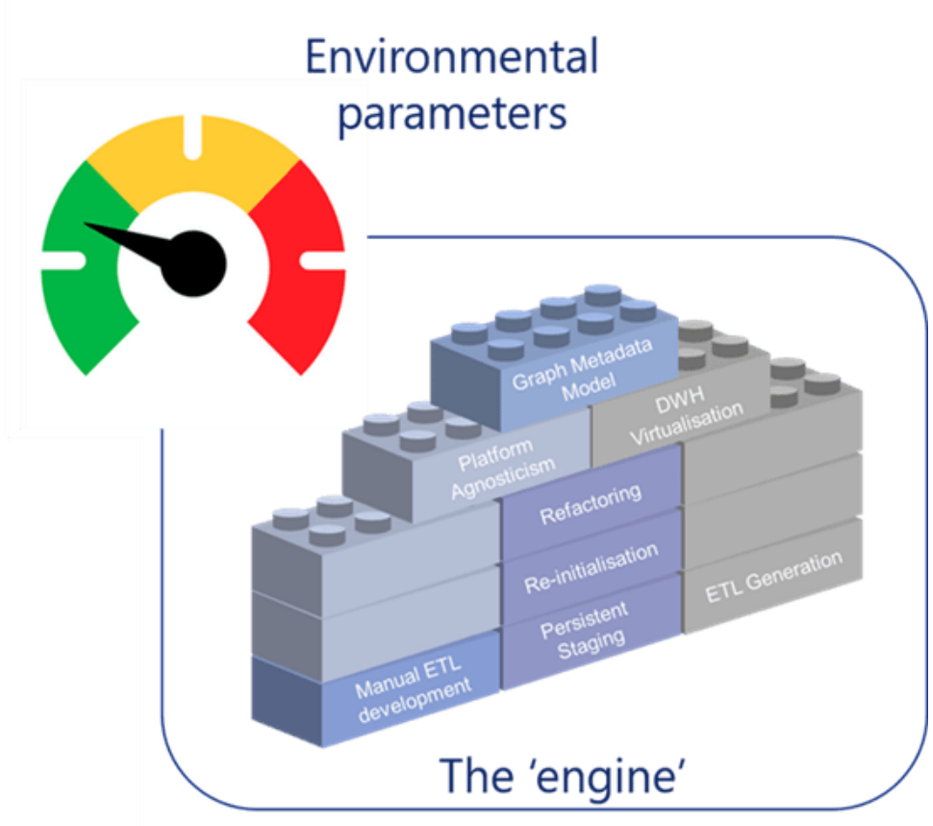
Data Vault inherently is a technique for physical design, so it stands to reason it will take more of a back seat.

The engine

I refer to this thinking as 'the engine', which I have outlined earlier and will touch on a bit later in this article as well. The engine is the combination of *design metadata* (concepts, patterns, ETL generation mappings) and *environmental metadata* (infrastructure parameters and constraints) that decides and manages how the physical models and corresponding data integration logic (ETL – Extract, Transform and Load) are generated, and updated when required.

Design metadata covers for instance what the core business concepts and units of work are, while environmental metadata covers the infrastructural and technological constrains such as available integration technology ('ETL tool') or available cores, memory and disk space.

The engine relies itself on various underlying concepts which are explained in more detail in my previous article on these topics but are outlined in the diagram below for reference. The premise is always that you work towards a mechanism that easily lets you model information, while automatically refactoring the solution in the background. In technical terms, this covers things like automatically redefining physical tables and reloading these with data using automatically generated ETLs (and much more).



It is this view that supports unifying the 'traditional' and 'new' data worlds; after all, certain things need to be done on the raw data no matter what. Do we need a Data Warehouse layer for 'Data Science'? Maybe not immediately and as a hard requirement, no, but it would be good if findings and data preparation outcomes would somehow be made available for other use cases. Similarly, it is helpful if already curated information can be used directly as opposed to reinventing the wheel.

The essence and vision on the interlinking of data use-cases in the context of enterprise information management is clearly and succinctly captured in Ronald Damhof's impressive Data Quadrant Model (read the article here, more information and generally on http://prudenza.typepad.com/). How can *Quadrant IV effort* be easily moved to *Quadrant II*?

This is what I'm exploring further here and this is where 'the engine', with its built-in Data Vault logic in the background, can assist.

An evolutionary view on Data Warehouse implementation

It seems that discussions on relevance of Data Vault often address very specific topics to make the case. For instance, 'do we still need Hubs?', 'isn't it better to create a separate table for each attribute?' or 'should I model this transaction as a Hub or a Link?'. These types of discussions are almost always focused on the physical implementation.

I look at using techniques such as Data Vault in an evolutionary way; how can we keep adapting the approach we use, keep the things that work in the current environments and move away from what doesn't make sense anymore. Having a certain degree of *atavism*, where certain traits reappear after being away for a while, is fine too. It's all about adapting to stay relevant in the context of your business.

There will always be some tension between the prescribed ways to use a certain technique, to be 'compliant' with an approach, and this kind of evolutionary thinking. By forcing yourself to stick to an implementation standard you will at some point reach its end of life (it will become extinct). However, by adopting a mindset of continuous improvement where the designs are altered on an ongoing basis, the chances of survival will greatly increase.

Data Warehouse Virtualisation as a concept is a good example of this. I use Data Vault concepts to deliver this, but the technical implementation itself has evolved significantly over time to meet new demands in both infrastructure and business use-cases. Ongoing evolution of the implementation now allows these techniques to span across (and using interchangeably) various different technical environments. But, while the underlying raw information remains the same, the upstream delivery including the Data Vault model has changed many times.

Just to be clear: Data Warehouse Virtualisation does not mean that everything always needs to be 100% based on views, it's the concept that changes in the model are directly and automatically translated into the required technology. It's refactoring in the extreme: if you change your model, your data morphs with it. How? By using the updated metadata to either recreate views, or by dropping and recreating the (persisted) DWH target table and regenerating and executing the corresponding updated ETLs to populate the target table again.

Why would you want this? Isn't the Data Warehouse model supposed to be relatively stable? My view is that this ideally would be the case, but that there is a maturity curve in play here. The early stages of implementation typically experience more volatility in the design, whereas in more mature environments – usually the ones with stronger information management & governance – the design tend to become increasingly stable. This is investigated in a bit more depth in an older post here, and of course varies from organisation from organisation. Generally speaking though, I don't believe we can have all the wisdom upfront to make every (modelling) decision correctly – including defining the right core business concepts even in the most basic Raw Data Vault approach. Data Vault supports more iterative approaches that lead to gradual understanding, as such the ability to refactor becomes important.

A second, more technical way, to look at this is again from the engine perspective. Automated refactoring takes away having to think about physical design decisions such as splitting Satellites because a few attributes have a high change rate. The engine may decide to create many physical tables that correspond to a single bit of context for a business key, or not – or it may change every now and then dependent on data dynamics.

Data Warehouse Virtualisation in this context also allows for versioning of one or more components of the design. I even see opportunities to leverage concepts from the Enterprise Service Bus (ESB) and Service Oriented Architecture (SOA) world in terms of versioning the Data Warehouse outputs (canonicals??) based on underlying schema changes. Using Data Warehouse Virtualisation concepts it's already feasible to host different versions of your Data Warehouse (or Data Marts) and allow subscribers of information to move to the newer version over a defined period of time.

Are we still talking about Data Vault in a physical sense? Not really; it's more about core business concepts, their context and relationships (unit of work). In 'the engine' I introduced earlier, this will be translated into physical or virtual Hubs (or not) and a balanced set of Satellites and Links – as well as the corresponding logic (ETL) to refresh these, if required.

You could argue this is still Data Vault, but at a higher level of abstraction. This would be true, although we won't be talking about Hubs, Links and Satellites anymore.

The point I am trying to make is that it's not particularly meaningful to focus on the physical implementation if you are deciding if a technique such as Data Vault is worth adopting. Rather, my recommendation is to focus on managing the required metadata to support your data solution and enable ways to keep adapting, stay flexible.

In terms of the delivery itself; this should be subject to some form of meritocracy and reason. If it works in certain niches, it's good. If you are following rules for the sake of it, that is not to anyone's benefit. Recent discussions on hash keys versus natural keys fit in this context. There are many ways to look at this, why not pick the best delivery for the specific use-case?

From mechanics to model

It is worth considering this: a few (approximately 5-10) years ago we were documenting the patterns to deliver a scalable physical data model. Now, we have a solid grasp on the physical model and associated loading patterns (both ingestion and delivery), which means we don't need to focus on the physical model as much anymore.

In other words, the mechanisms of the Data Vault patterns are becoming a commodity, and as a result can be automated and are losing visibility. They are being pushed 'into the engine' of data management. This does not mean that the concepts are no longer valid; it just means we will start accepting this as a standard and don't need to think about them as much.

As I stated earlier, there seems to be little point arguing if a Hub needs to be persisted or whether we should use hashing or not. This can all be automated, so we have time for more value-added work such as additional emphasis on information modelling.

Spaghetti code

No matter what technology you have, there is no escaping the necessity to model information – to understand the concepts, relationships and behaviour of data.

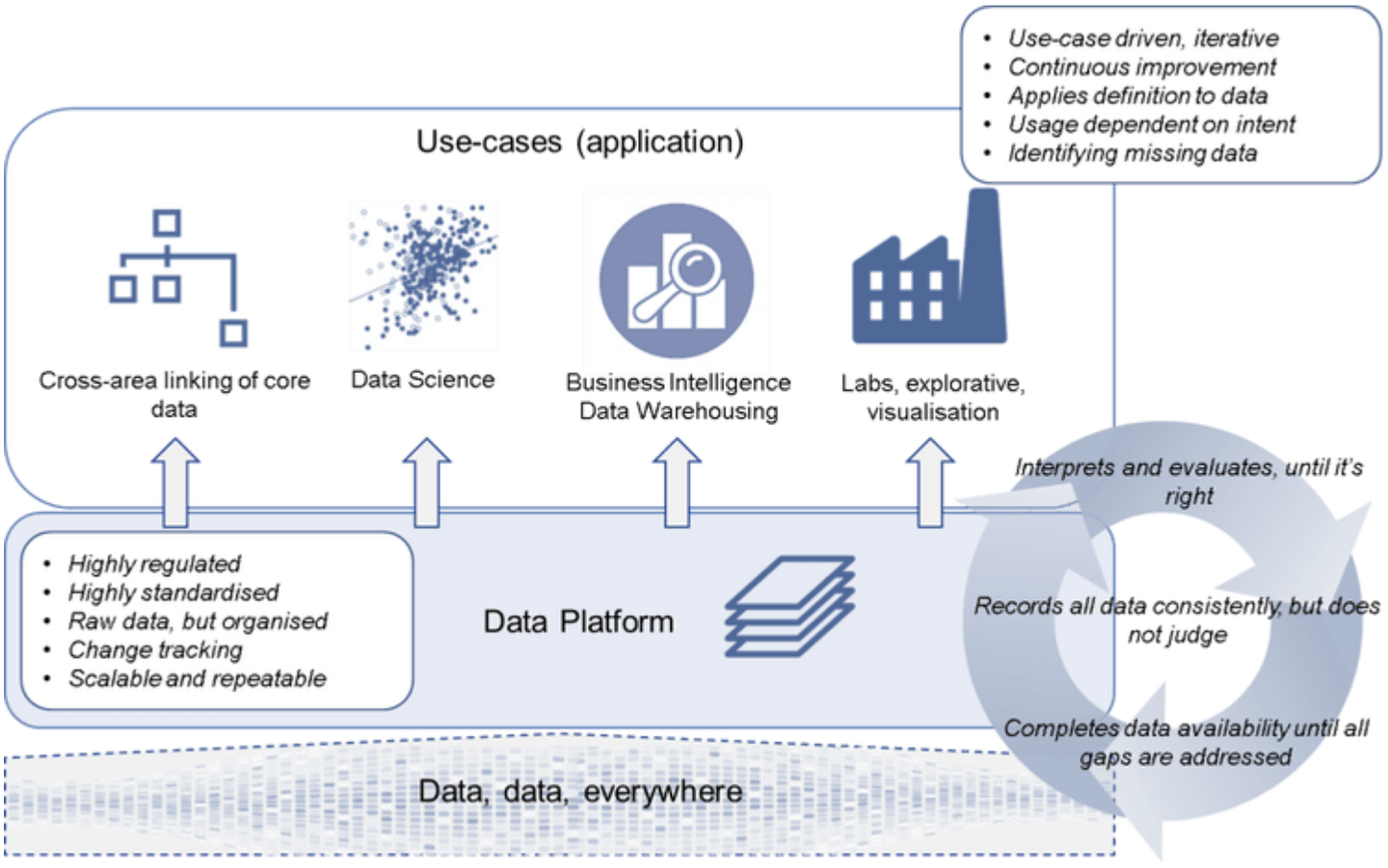
In areas which heavily focus on outputs using raw data, which is common in the traditional Data Lake thinking, this often results in relatively large and complex code segments. The code also needs to manage clean-up, rules and fundamental time-variance logic.

This very quickly results into a spaghetti-like mess of tangled code segments. This is one of the topics that Ronald Damhof touches on in the context of his Data Quadrant Model: how to move insights into a production state, so benefits can be realised on an ongoing basis (moving from Quadrant IV to Quadrant II).

I'm not saying it is wrong to pursue this arguably more explorative use of data. On the contrary, there is definitely value in this (which is also acknowledged as Quadrant IV in Ronald Damhof's model). What I'm saying is that this does not replace the more structured management of data as is implemented in a Data Warehouse architecture.

In fact, it is quite easy to combine these. In my designs I have done so by incorporating the Persistent Staging Area (PSA) concept into the architecture.

Following this idea, you can support both the explorative and structured ways of working with data, as is displayed in the diagram below (the Data Platform being the PSA).



As is often the case, technology is not the answer – but a flexible approach and proper metadata management is. I believe we will get the best overall outcome if we keep simplifying the way we can perform information modelling and continue to abstract out the ever-changing technical delivery.

Imagine you could quickly organise business concepts and context without having to worry about a physical model and create a point-in-time set around these core entities in a scalable environment. Having all the separations of concern built-int? Wouldn't that be much easier to code against using Python for data science purposes (if that's what works best for your specific use-case)?

I believe it would be, as a large amount of fundamental data management logic is already catered to, and any reusable logic can be 'plugged-in' the available set (technically for instance as a Business Data Vault object – but that is happening automatically in the background) – where it will be usable for other business areas, too.

This means we're still using Data Vault, or equivalent hybrid techniques, but the patterns and ideas have blended in the background to enable simplification of design and development.