The promise of Data Vault is that it is always **auditable** and to guarantee that auditability data vault also promised you that you do not need to **refactor** the delivered data vault architecture while remaining **agile**, **agnostic** to platform and embrace **automation**.
Let's examine these anti-refactoring demands by exploring how data vault delivers on them.

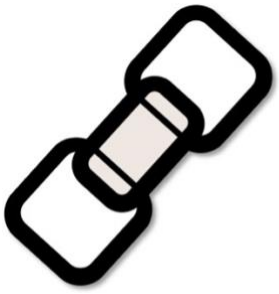
Demand #1: When I have a new business requirement, I do not want to have to reengineer what I already have

Data analytics has grown as a discipline over a few decades and the two competing data warehouse modelling paradigms have been Inmon and Imhoff's 3rd normal form Corporate Information Factory (CIF) and Kimball's Dimensional Modelling. CIF is highly normalised with limited redundancy bringing data from disparate sources into an integration layer of the data warehouse. Dimensional models are instead denormalised into facts (measures and metrics) and dimensions representing slowly changing facts for those metrics. The long-standing issue has been the rigidity of these structures, in other words as business requirements evolve and the business itself grows, making changes to the schema has become costly, leading to legacy environments and the cost of the analytics system outweighing its benefit. CIF and Kimball are not easy to change or integrate with new business rules and processes and remaining auditable; this in turn often leads to **migration** efforts.
In short, Kimball and CIF style data models ***do not scale***

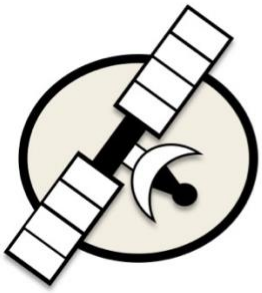
Data Vault's response:

The Data Vault model is designed to change, only then can it truly last. Because there are only three table types (hubs, links and satellites) an overall data vault model can be delivered in iterations, and every change is really an addition to the overall model. Let's get familiar with the table types.

"the effective centre of an activity, region, or network" - Oxford
A **hub table** contains a unique list of entities or things that represents a domain or concept within the enterprise. Hub tables act as the central immutable store representing business entities, everything else connected to the hub gives us more context about that business entity whether that be its relationships to other entities or descriptive information about that entity itself.



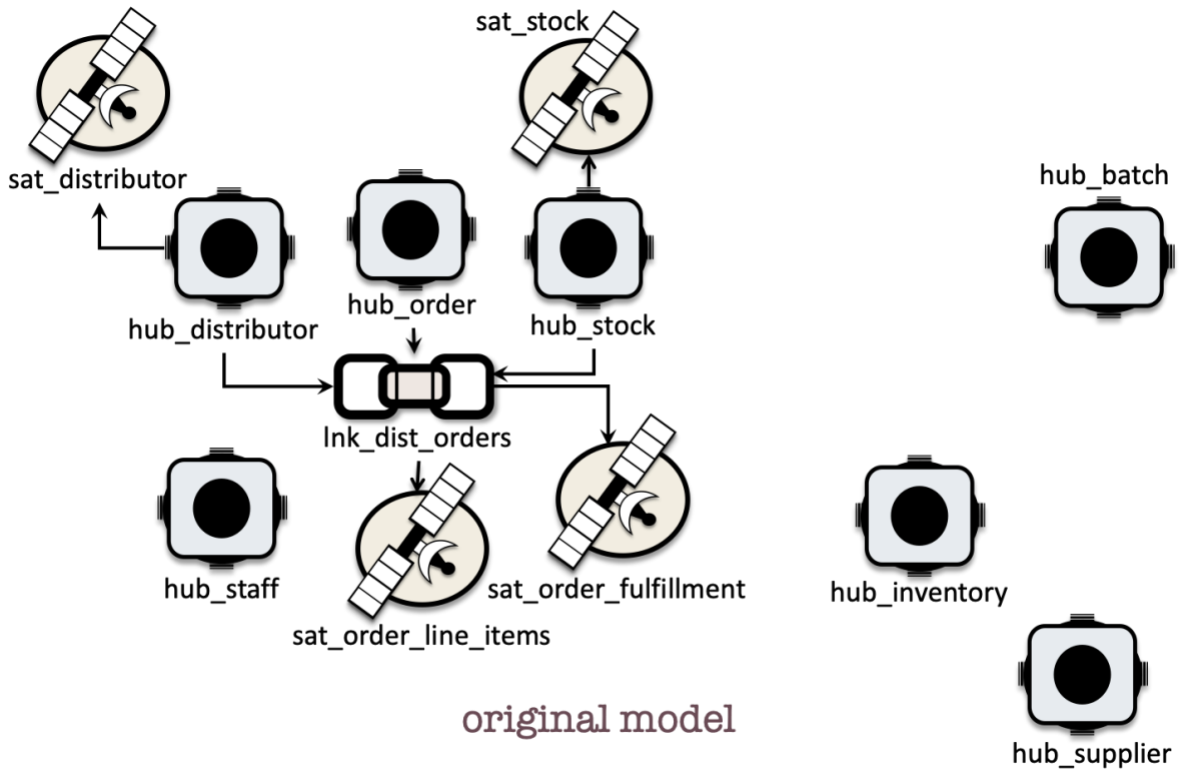
"a connection between two or more people or things" – Oxford
A relationship is supplied in its raw form from application sources as a business key relates to one or more business keys either between different domains or within the same domain or even across source systems, these are stored in a **link table**.



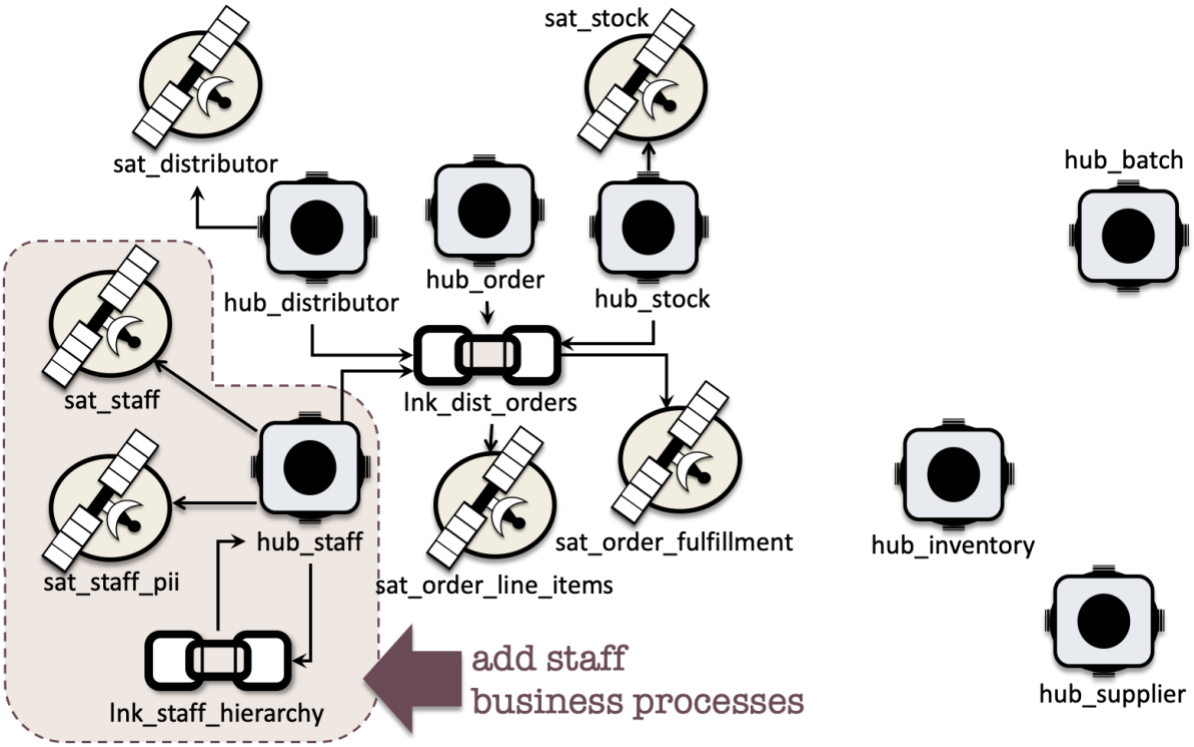
"something that is separated from or on the periphery of something else but is nevertheless dependent on or controlled by it." – Oxford
A **satellite table** contains the descriptive details about the artefact it is adjacent to. If the satellite is adjacent to a hub then it will contain the change-tracked descriptive details about a single business entity in a **hub**. If the satellite is adjacent to a **link**, then it will contain the change-tracked descriptive details about a relationship in a link.

When you model your data warehouse it should align to what you understand is your **business objects** as represented in a **business architecture** model. Let's use an example from an existing article about beer (see: bit.ly/3wZbGql)

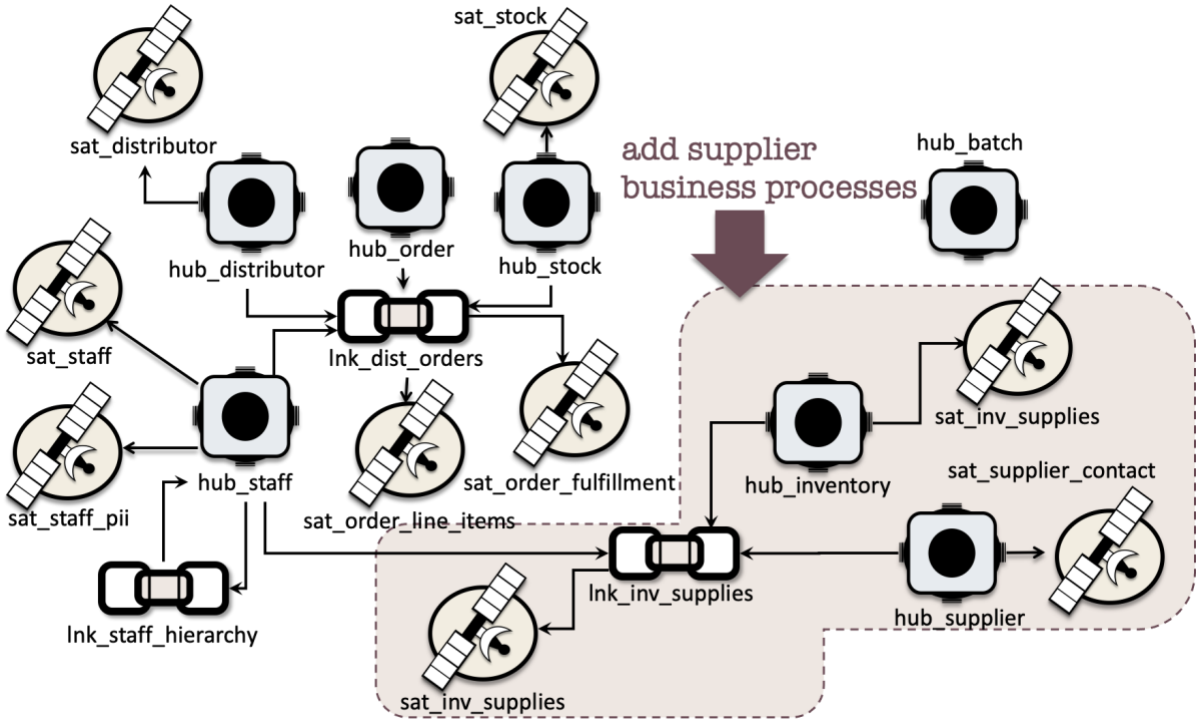
At this moment we might not know how to model the rest but that's the intention of Data Vault deliver, it is **agile**, just like the model itself.



The adage is true in data vault, don't boil the ocean! An enterprise might now know what their complete business architecture is, but the business will have a good idea. With the lack of a business model (represented as an ontology and taxonomies) and as a part of data vault model we will end up building one! As new business cases arrive, we begin to model the entire data vault! From the original model, some sprints later we start to add more business requirements and inevitably more to the business data model.



Not forgetting that data vault has a specific purpose, aligning the data warehouse to the business architecture. The need to query the data is resolved using dimensional marts on top of the data vault model. The data vault is not exposed to the BI tools, instead the information marts are. And because the audit history is solved in the data vault, the information marts themselves are now **disposable**.



You can see, Data Vault scales!

Demand #2: When I integrate a new data source, I do not want to reengineer by integration points

To understand this demand is to understand where integration happens in the data vault, the hub tables. There are some things to consider,

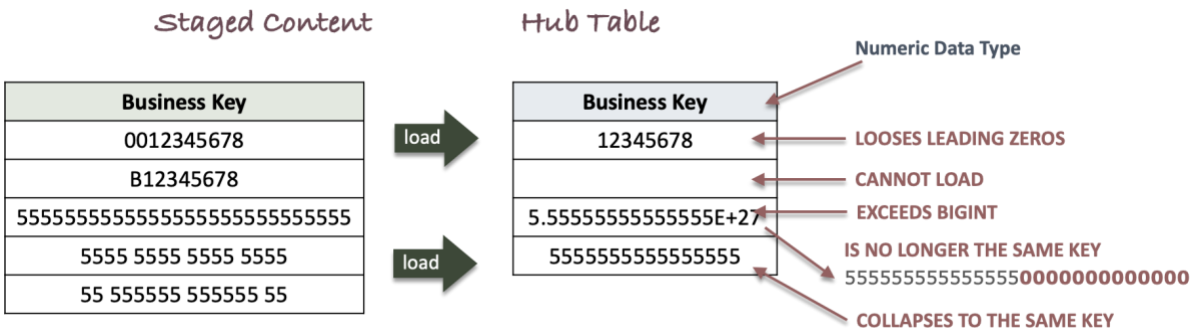
- like the same immutable key shared across source systems to fulfil the business process, that represent the same business object
- similar business key formats being loaded to the same hub table but represent different business objects.

We deal with the latter using business key collision codes, see: bit.ly/3xlFK0s

Data Vault's response:

Business keys come in many forms, some are called **keys** in the source, **ids** or **numbers**. No matter what they are called in the source providing that business object's immutable key they must always be stored as a **character data type**, *period*. That integration point in data vault is the business architecture aligned hub table. A hub table can be multi-source, and account number in one source system may be called account id in another and they are **identifying** and **not** a value that benefits from being stored in a numeric data type.

Should a length of a character data type be insufficient it is far easier to extend the length than changing data types, for those business keys already loaded what is the data loss? And worse, what integration technical debt had to be paid in order to make sense of the business key in reporting. If a new source is added to the same hub table, are we then forced to **refactor** the hub table? No, load the business key as a character, always. Let's see what happens when we do use a numeric data type.



If Data Vault meets this demand there must be discipline applied, the model must not leave it up to the presentation layer to resolve integration debt. In other words, never concatenate keys to squish account keys into a single hub. If you have a hub_account with a single business key like account_id, a bank account should go to a hub_bank_account that includes a composite key, branch code + account number. If you leave it up to the consumption layer to differentiate that means the same logic to break up the concatenated keys must be applied in every query, unnecessary.

Demand #3: When my source data evolves, I do not want to rebuild my data model

Businesses grow and so do the information about the business processes. Some even update the same source file that is landed to be ingested by the analytics platform, you definitely will expect that this could be the case with semi-structured data. What then do we do when new content arrives in the business process? Do we keep refactoring the data model?

Data Vault's response:
Know your data. A new column changes the grain of the data, but it does not automatically mean we add the columns to a satellite table, what are they? Where does that column belong? This is under the category of **schema drift / evolution**. In the article "Data Vault Test Automation" (see: bit.ly/3dUHPIS) we discuss **record condensation** in relation to **satellite splitting**. In addition, if the new addition is not pre-empted, we need to know:

- is it a new business key?
- Is the column personally identifiable?
- How does the new column effect the grain?

You need a process to:

- Detect the new column(s)
- Profile the column(s)
- Map it
- Continue processing

But under no circumstances do we need to reload the data in the data vault, we do not refactor!
Here's why.

LOADATE: 2020-10-21

Staged Content

First Name	Last Name	DOB	HashDiff
Michael	Fox	1961-06-09	a12abc
Jeffrey	Weissman	1958-10-02	bb7ac1

sha1_binary('Michael'||'Fox'||'1961-06-09')
sha1_binary('Jeffrey'||'Weissman'||'1958-10-02')

load

Satellite Table

First Name	Last Name	DOB	HashDiff
Michael	Fox	1961-06-09	a12abc
Jeffrey	Weissman	1958-10-02	bb7ac1

LOADATE: 2020-10-23

Staged Content

First Name	Last Name	DOB	HashDiff	Middle Name
Michael	Fox	1961-06-09	cba99	Andrew
Jeffrey	Weissman	1958-10-02	dd18d	

sha1_binary('Michael'||'Fox'||'1961-06-09'||'Andrew')
sha1_binary('Jeffrey'||'Weissman'||'1958-10-02'||'')

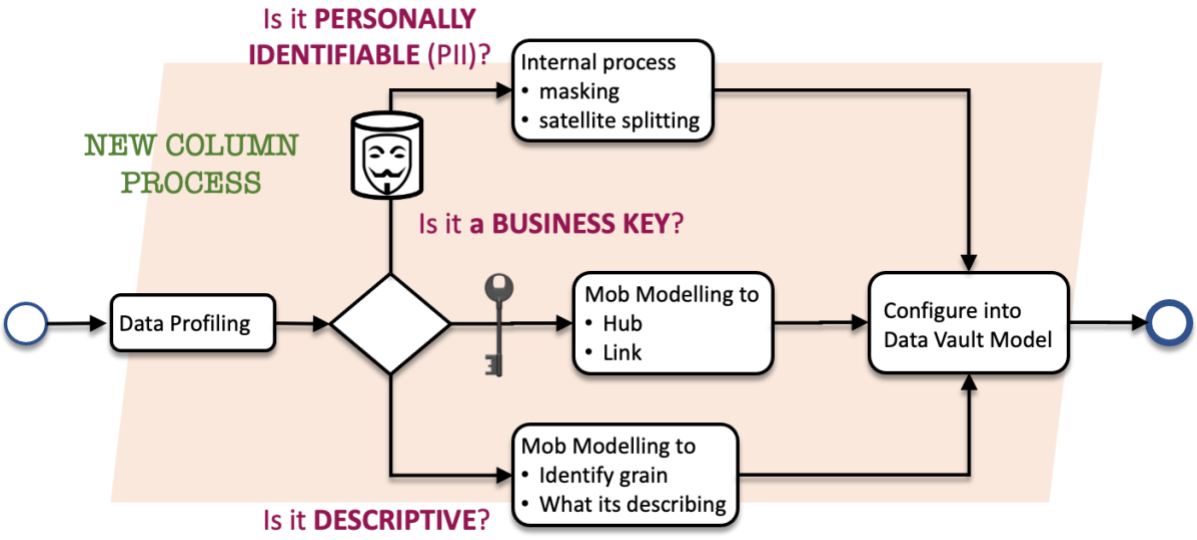
load

Satellite Table

First Name	Last Name	DOB	HashDiff	Middle Name
Michael	Fox	1961-06-09	a12abc	
Michael	Fox	1961-06-09	cba99	Andrew
Jeffrey	Weissman	1958-10-02	bb7ac1	
Jeffrey	Weissman	1958-10-02	dd18d	

The shape of the satellite table used for reporting is auditable, when the new column was added before 23 October the next load will include the new column 'Middle Name' in the HashDiff calculation and therefore in itself probably insert a new record for every **current** record in the target satellite table. And that is good, in essence it has completed a migration in itself! Before 23 October we did not have a concept of a 'Middle Name', on 23 October we do. Because the hashdiff now differs the new data is loaded. The fact that Jeffery does not have a middle name is a **different** fact to the data model (represents the business process) not having a concept of 'Middle Name'. Therefore, by adding to the existing model the way we did above **does not create duplicates** in the satellite table and we do not need to regenerate hashdiffs for existing data. It's unnecessary.
How do we identify this ***change in fact***? Metadata of course! Reconciliation tracks the number of columns in the satellite table historically. (see: bit.ly/3dUHPIS)

Here's how.



Ideally the analytics team should be told when new columnar data (or key values in semi-structured data) arrives in existing data pipelines, but just in case the data pipeline itself should have a schema drift detection mechanism to kick-off the above process.

Demand #4: When a correction is needed, I want to remain auditable and keep correction costs low

Murphy's Law dictates that **anything that can go wrong will go wrong**. That is true in engineering and certainly on data! Therefore, of everything you build you must include a plan to deal with failure. In data this can be particularly costly,

- Corrupted data must be reversed, and the corrected data replayed and reloaded
- Depending on how widespread the corruption occurred, downstream information marts and reports must be corrected, and stakeholders informed on the incident.
- Data that arrives in the wrong sequence usually means that the existing data must be reversed to the point the out-of-sequence data is applicable and subsequently reloaded.
- Depending on the volume of the corrections to downstream calculations because they were not idempotent may take weeks and months to correct, how then can we trust the data?

Data Vault's response:
Out of sequence data is a real problem to data analytics, for streaming content the issue is dealt with ever improving streaming tools like Apache Beam, Flink, Kafka and more and more new algorithms and frameworks are being developed to deal with this. For batch loading scenarios Data Vault has a Data-Driven loading pattern to deal with out of sequence batch data, the eXtended record Tracking Satellite (XTS). This pattern means there is no intervention from the analytics team to correct the timeline themselves. And because information marts are ideally delivered as views over the data vault, no need to rebuild them either. See: bit.ly/3wZbGql for more details. Note that this pattern is only possible because of:

- Applied Date, explained in the article link above, and
- Because Data Vault 2.0 has no END-DATE column

The other Data Vault pattern is the use of reconciliation, yes because data vault is based on repeatable patterns so too is the reconciliation framework described here: bit.ly/3dUHPIS What's more each record in data vault can have the Jira id column attached, what's guaranteed in an agile enterprise is that agile **Jira stories, epics and tasks** are always documented. Why not include that metadata in your Data Vault model?

In conclusion:
For too long data analytics teams had **to foot the bill** for refactoring any of the scenarios described above. Data Vault has the answer because it **scales** and has the **repeatable patterns** that make it easy to maintain. It is a new way of thinking about the data analytics (data warehouse and lake) and thus before jumping into a data vault project after learning what hubs, links and satellites are, take the time to understand its structures, why the math behind it just works, automation and how to consume from the data vault model.
Get a copy of "the Data Vault Guru" (amzn.to/3d7LsJV, amzn.to/3nsqTfR, amzn.to/30IxOYF), read about what is in it before you buy: bit.ly/3bcayaO and check the Amazon review comments too!

The views expressed in this article are that of my own, you should test implementation performance before committing to this implementation. The author provides no guarantees in this regard.