Seven Deadly Sins of Fake Vault

Seven Deadly Sins of Fake Vault

The term “Fake Vault” was coined at wwdvc 2019; Fake Vaults are supposed Data Vault 2.0 implementations that fail the basic tenets of an auditable, agile and automated data vault model. They include ingenuity that does not scale, that in fact flies in the face of what a scalable data vault model should be. Terms like “bag of keys” or “keyed-instance hubs” are not Data Vault 2.0 terms!
In no particular order here are the seven deadly sins of fake vault:

- ***GULA - "Business Vault is the layer exposed to the business"*** - the implication is that Raw Vault is not exposed to anyone and Business Vault is the "conformed" business layer with column names in the form digestible by the business. This is an **incorrect** implementation of Business Vault. Business Vault is nothing more than the archived outcome of derived business rules **based** on Raw Vault. It is ***not a copy*** of Raw Vault; Business Vault ***is sparsely populated*** that serves as the implementation to resolve business process ***gaps*** in the source application. For example, a 3rd party source application may define that a home loan account has its own facility and an offset account may have its own facility too but the business only sees one facility that is an umbrella business entity for all related accounts (loan and offset accounts). The raw vault link will contain how the 3rd party source defines this relationship but the business vault link will have the business view of that relationship. It is not a replication of the source but fulfilling the **business view** of the business process. Additionally, like raw vault, the business vault is **auditable** and **reuses** the same loading patterns. Another use (in addition to plugging source-system gaps / defining a different view of the source) may be to centralize and create an audit trail of derived content that is automated and closely tied to ***data governance practices***. It may also serve as a ***temporary store*** of derived content until the raw source can supply that content instead.
  BV is based on RV, ***RV+BV=DV***.

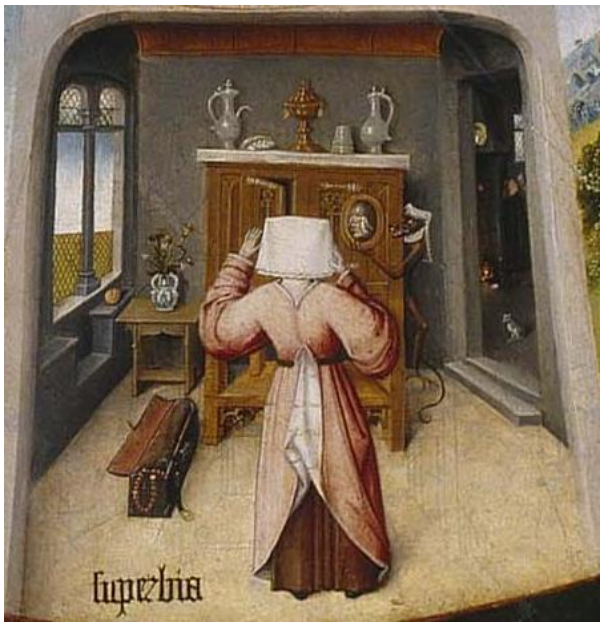
- ***ACCIDIA - "Automatically decomposing Units of Work (UoW) into a bi-relationship link table although the Unit of Work had more than just the two business entity composition".*** Decomposing such a relationship between business entities into two-part link tables breaks the UoW and in many instances may make it impossible to recreate the source if needed, a fundamental requirement for data vault, **auditability**. Besides why break the UoW only to have to recreate it downstream? Not only does this add latency to time-to-value but it also becomes costly in terms of query execution to break the UoW up and reassemble it; but also increases the number of tables to join which makes the data model solution perform poorly.

- ***LUXIRIA*** - Creating ***Weak Hubs***, these are Hubs that do not track a unique list of business entities; or another definition is using too generic hub table names. The former occurs when we identify keys and assume that these should be loaded into their respective hubs to create a link to them when in fact such keys would be better defined as **Dependent-Child keys** in a link or satellite table. The so called "weak" key doesn't describe anything on its own and needs a "strong" key in order for it to have any meaning to anyone using that key. This is <u>exactly</u> what a dependent-child key is! The latter, using too generic hub table names and definitions relates to overloaded content landed and that needs to be modelled. We have seen this, a single business key column that has multiple business entity types within it and a decision is made to load a ***hub_entity***. The name itself doesn't convey any meaning and if you take a step back and think about what you have just created is a ***dependent-child hub!*** Without something to better define that hub key you need a code or category to better define the business key! The overloaded business key column must be split either by (in order of preference):
    a) the source to provide the content split as independent files/tables to be staged or
    b) we have to split the content before staging.

  The former is preferred because it solves the integration debt upfront and the task of keeping this up to date and managed is with the ***source application***. The latter means there must be additional code to split the content and the need to build *error*

traps to catch the moment a change is detected in the rules to split that column. An overloaded column can also mean a need to implement ***row-level*** *business key collision codes*, more maintenance in your data warehouse!

- ***SUPERBIA*** **- Source system data vault (SSDV)**, a key understanding of data vault is the need to integrate multiple business processes (and their outputs) by their ***natural keys*** which may be served by multiple source applications. If source applications share natural keys then the *passive integration* naturally occurs and the entire business process is tracked in harmony in the data warehouse. A great example is by treating transaction-style MDM as a source because its harmonized (match-merged) key is used to integrate multiple applications in **realtime**. No collision of business keys can occur (a single key value that could represent multiple business entities within the same hub table). BUT as diverse as the business decisions are to purchase specific software to automate various portions of the business service, integrating the business processes into a single data warehouse will likely come across where such collisions to a single hub table could occur, only then do we use a business key collision code (BKCC) to ensure that loaded business entities remain ***unique*** within that hub. To use a collision code based on the source-system that provided the key, without considering that two or more source applications could be using the ***same*** business key value to represent the ***same*** business entity leads to a source-system vault, an anti-pattern. You will inevitably create more entities than you need to the hub table, more joins then required to bring the content together and more tables required in business vault to solve the *integration debt* you would have introduced if you followed this source-aligned loading pattern. Also remember that the appropriately applied *business key treatments* is performed (left+trim+upper) **before** hashing.



- ***IRA*** **- Staggered loads.** Surrogate *sequence* key-based data vault needed to follow a similar loading paradigm as dimensional tables, first all the dimensions must be loaded before the fact table can be loaded, sequence keys in a dimensional model implies temporal meaning from each dimension table to the fact table. For Data Vault before becoming based on hash-keys, the same was true, you needed to load hub tables before loading hub-satellite and link tables, you needed to

wait for the link table to be loaded before you can load a link-satellite table as each step needed to *lookup* the parent key in order to maintain referential integrity (foreign key constraint), although the sequence key here never had any temporal meaning, they were in fact *retained* keys (one-to-one relationship to the natural key). This led to loading paradigms where loads were dependent on each other and could not continue until its parent table had completed its load. Some implementations have gone as far as loading all hub tables first, followed by all link tables, followed by all satellite tables. So much time is wasted (muda) waiting for table loads to complete that you could only really get away with this loading paradigm in overnight batches! Surrogate hash keys meant that after the content has been staged that all data vault target tables can be loaded *independently* and still guarantee referential integrity (RI) either as post load automated orphan checks or as *deferred* RI. Each portion of the enterprise data vault is ***eventually consistent*** ensuring that the data does flow and does not need to wait for any other portion of the data vault model to complete its load before its own load can continue.

Surrogate hash keys were used because:

a) they provide a more random-looking key value to spread the data on a MPP platform that also ensures similar data is co-located on disk, and
b) they make joins between data vault artefacts simpler by ensuring only a single column is needed to join be between these data vault artefacts (rather than using natural keys which may include the BKCC and composite keys)

- ***INVIDIA*** - Not recognizing ***Link-Satellites***. As an extension of the 3rd point above, a data vault model without link-satellites lacks the descriptive depth to describe a relationship. Instead in these "data vaults" a hub table is created simply to support a hub-satellite that describes the relationship. That hub table is what we want to **avoid** because it is not by (Oxford) definition a hub and it just adds more tables to include in the join, thus affecting performance to get the data out and having yet more staggered loading. In recognizing link-satellites also recognize that even if source data is supplied with multiple keys it does not mean that all the content must by default go to a link satellite. If all or part of the content is describing only one business entity then that content must be mapped to a hub-satellite table (or split between multiple hub-satellite tables and a link-satellite tables), thus loading content to what it is describing through **satellite splitting**. Not performing satellite splitting forces the consumption of the data vault into queries that have to "re-envisage" descriptive-content change ***in-flight*** every time that content is sought. In other words to get a business entity level-grain out of a link satellite table you have to execute a “select distinct” by the business key of the relationship in order to get to that grain ***every time***. Instead satellite splitting ensures that content is already appropriately propagated at the grain you need and that a change record is a ***true change***. On the subject of satellite content, an old DV

motto is *"all the data all the time*" and it still rings true. If the source supplied 600 columns, do the due diligence and split the content appropriately so that you can recreate the source and remain ***100% auditable***. For performance and regulatory reasons split the satellite by personally identifiable information (PII), rate of change and by clustering together critical data elements as well as the other satellite splitting reasons listed above.

- ***AVARICIA*** - Using anything other than ***string/varchar*** for recording business keys in hubs. Creating hub tables with integer/numeric data types simply because the business key has the word "number" in it does not mean the column is a ***measure***. Measures are additive/semi-additive fields that are appropriately data typed to suite performing algorithms on those said columns. Business keys are immutable, remain consistent for the life of the business entity and are not values that arithmetic should be applied to. And even if you do you have now created a ***new key***. On top of that ensuring that the hub business key column is of type string/character/varchar ensures that the hub table will ***least likely*** ever need to be ***refactored*** and that you can load any other data source to that column ***without*** breaking the hub table. On the subject of data types, surrogate hash keys must always be delivered as raw/binary data types and all joins are performed in-database and ***not*** in the BI tool, especially those tools that are not built to handle complex joins and binary data types!

Images from "The Seven Deadly Sins and the Four Last Things" by Hieronymus Bosch, bit.ly/3phFHh6

Four Virtues of Data Vault 2.0

Four Virtues of Data Vault 2.0

- **(PRUDENTIA) Agile** – not only is agility a virtue of Data Vault in terms of data modelling, but also in delivery as well. First off, changes to the existing data vault model are non-destructive but rather they are added as additional outcomes from business processes. Data Vault also support **schema evolution**, so if your data source happens to have additional columns coming in then these columns are added to the end of a satellite table; depending on what it describes. Deprecated columns are simply no longer populated but **never discarded**. The agility in the overall data model is further illustrated in its ability to support *disposable* Kimball dimensional data models! Yes the Data Vault 2.0 does **not** replace Kimball, the objectives are different! But rather the Data Vault is modelled to deliver a **Data Warehouse** that in turn supports **virtual** Kimball data models. PITs and Bridges (query assistance) are provided to help with performance as your **enterprise** data vault model scales **horizontally** with new business requirements and data sources (and business vault) the data being loaded to the data model scales **vertically**. This also means that **independent agile teams** can add to the enterprise data vault model without impacting each other and the practice itself promotes collaboration through standards and governance. Raw vault records the data outcomes of source application raw data, business vault records the data outcomes of derived content based on that raw data; the two live together but are decoupled giving further agility to the overall data vault model.
- **(IUSTITIA) Automation** – rather than conforming data into a bespoke dimensional mart a data vault is delivered with automation of just **three** loading patterns:
    - Unique business keys are loaded to business ontology (or subject area) **hub** tables.
    - Unique relationships / units of work are loaded to **link** tables, these depict the unique relationships that business entities have to each other as business processes dictate.
    - Descriptive content about a business entity or a unit of work are loaded to **satellite** tables

  This means that once the automation patterns are established the cost to model more data sources and derived content to meet your business requirements reduce and the cadence to delivery increases.

- **(FORTITUDO) Auditability** – yes because we have not conformed any of the data the content remains auditable. I can recreate the source file at any point in time, a key requirement for auditability. As a part of the automation patterns each and every data vault artefact includes a standard set of DV-metadata tag columns to show where the data came from, when it was loaded and what is the applied date of the set of records. You may choose to add record level metadata tags as you see fit, which may include the job or task id of the job that loaded that record, Jira id of the project or task responsible for loading that record or even the user id used to load that record. The same auditability applicable to raw vault is expected of business vault; after all business vault is just the recorded outcome of business rules based on raw vault; raw vault is just the recorded outcome of business rules based on application sources.
- **(TEMPERANTIA) Agnostic** – to platform, data vault conceptually can be built on SQL Server, Teradata, Oracle, Snowflake DB, Hadoop & parquet as a Hive store, Redshift, Azure Synapse, it does not matter. A part of the implementation decision falls on the delivery platform. Surrogate hash keys were developed and deployed to take advantage of massively-parallel processing's (MPP) distribution of content based on a key. If the platform of your choosing does hashing already based on a nominated key then does it make sense to hash the business key yourself before it is internally hashed again? It might depending on your preference! Not all business keys are represented by a single column but might be represented by a composite key, let alone the use of a BKCC would need to be included if hash-keys are not used! What if the data vault model is also multi-tenant? That tenant id with a BKCC and then with the business key column(s) would make up the hash key, your non-surrogate hash key based data vault model would need to include those columns in every join!

Images from "The Cardinal Virtues" by Raphael, bit.ly/3dbEwqQ