Contents

Data Modelling, Data Vault 2.0 and Data Mesh

Data Modelling

> Data modelling is the process of analysing and defining all the different data your business collects and produces, as well as the relationships between those bits of data. Data modelling concepts create visual representations of data as it's used at your business, and the process itself is an exercise in understanding and clarifying your data requirements.

What is data modelling: MS Power BI

> A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities. For instance, a data model may specify that the data element representing a car be composed of a number of other elements which, in turn, represent the color and size of the car and define its owner. The term data model can refer to two distinct but closely related concepts. Sometimes it refers to an abstract formalization of the objects and relationships found in a particular application domain: for example the customers, products, and orders found in a manufacturing organization. At other times it refers to the set of concepts used in defining such formalizations: for example

> concepts such as entities, attributes, relations, or tables. So the "data model" of a banking application may be defined using the entity -relationship "data model".

Data Model : Wikipedia

> Data modeling is the process of creating a visual representation of either a whole information system or parts of it to communicate connections between data points and structures. The goal is to illustrate the types of data used and stored within the system, the relationships among these data types, the ways the data can be grouped and organized and its formats and attributes.

IBM : Data Modelling

Observations

- Data modelling and data models are foundational to all information systems.
- Data modelling brings a deeper understanding to data.
- A data model assists reasoning about, and managing data.
- Well modelled data greatly increase the data's value.
- Properly modelling data upfront sets the data-intensive project up for success.
- Failing to properly model data early on in the project introduces a lot of risk and unknowns - data debt.
- Per definition, a data model has a fixed scope and can only model so much, and this is acceptable and even desirable since it narrows focus to what is really important.

Profiling raw data

The most important task in any data modelling project is profiling the data in order to develop a sufficient understanding to drive the modelling. Without a sufficiently complete comprehension of the data, it is impossible to model the data effectively. Profiling the data involves extensively exploring the data's schema, types, typical values, relationships and identifying unique keys. This is most often performed through querying raw data using SQL, Pyspark, Excel or any other tool that allows for rapid querying. Although you might be so fortunate to have other documentation at your disposal explaining the data, you still need to query it to verify all assertions.

> Give yourself enough time to properly and thoroughly understand your raw data! You will not regret the time spent.

About Data Vault

Essentially, a "data vault" is a layer between raw data and purpose built data warehouses. We have these challenges to overcome:

- we do not want to tamper with raw data because our tampering might be faulty, yet we want to be able to easily revisit raw data
- reprocessing everything from raw source is too complicated and expensive
- we want to keep track of changes in the data (CDC) in a storage technology independant manner
- we want to capture and encode what we have found out structurally about our raw data so that everyone else can benefit from this understanding
- because we ingest frequently we want to optimise for fast, cheap and scalable insert

Peter ter Braake, in his book "Data Modeling for Azure Data Services" puts it this way:

> The general idea is to build a layer between the data's origin, the line-of-business applications, and its analytical use in a dimensionally modeled data mart that is optimized for specific usage. The data warehouse is the layer that should be stable over time. When processes or legislation change, the data itself does not, so the data warehouse should not. In that way, you create a repository to store historical data. The repository, the data warehouse, should be optimized for the long-term storage of historical data. It should be flexible in that when new types of data are introduced, new data should be easy to add to the data warehouse. By now, we know that when you optimize any type of database for a certain use case, it probably is not optimized to do something else. A Data Vault data warehouse is optimized for the long-term, flexible storing of historical data. It will, most of the time, show really bad query performance for the types of queries that we write to create reports. It is not made for using the data at all.

- Data vault modelling is well suited for OLAP ingest, but poorly suited for any kind of querying.
- A data vault needs to be augmented with business / data marts: mini data warehouses in the Inmon and Kimball tradition.
- A data vault is agile and can evolve:
  - You do not have to specify it all upfront, but can develop and extend it as time goes on.
- It is not a Data Lake, although you could implement a Data Vault using Data Lake technology.
- In general, it prefers implementation in a cloud anlytics type of database, with column store physical table layout.
  - storage and compute tehcnologies such as Snowflake Computing's main offering, Google's Big Query, Amazon's Redshift, Databricks

SQL Warehouse are good choices.

- Can be thought of as evolution of Star and Snowflake Schema modelling for analytics.
- modelling data to optimise or achieve:
  - source system independence
  - longgevity: to outlive current source system specifics
  - rapid ingestion at high volumes
  - supporting parallelism in the ingest phase
  - tracking changes over time
  - as a source from which further analytics and data science can happen
  - as a means to standardise and simplify how analytical data is modelled
  - as a means of persisting data in a raw, but useful format
- Introduces a layer between completely raw, and transformed data from which future transforms can happen.
- Aims to push transformations as far down the line as possible, to a single stage of the data life cycle.
- Combats the tirany of figuring out what transform happened where.
- Enable "straight join" ability via the use of ghost records, greatly speeding up joining.
- Designed with big data use cases in mind.
- Designed to be source-system agnostic: your data will live far longer than this or that source system, so do not depend on any keys from it.

Data Vault Contrasted Against Straight to Data Warehouse

Essentially, Data Vault does not aim to replace traditional Data Warehouse modelling wisdom, but rather pitches itself as a layer between raw source data and the Data Warehouse. One of the goals of a Data Vault is to provide the bedrock from which to build many currently unforseen domain and business specific Data Warehouses and data surfaces in a repeatable manner. This is tipified by the realisation that at any point in time, we more completely know what data we have than what questions the data in our hands might be able to answer in the future. Driven by this realisation, we build a structure that should be able to answer combinatorially more future questions than simply transforming data for a specific set of end-user use cases.

Additionally, Data Vault strives to capatalise on the new features of distributed cloud analytics and massively parallel processing (MPP) storage systems. Data Vault prefers an extract, load, transform ELT process where transformation is encouraged to happen inside the storage technology. Contrast this with the traditional external transformation paradigm that requires data to be read into a compute system such as Spark, transformed, and then written back. A major theme of Data Vault is to make it easier and easier to build out denormalised views on data, driven by the realisation that storage is currently far cheaper than compute.

Traditional Data Warehouse

- A subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making.
- A specially prepared repository of data : the data has been wrangled much and possibly altered the meaning of the original, raw data.
- Composed of facts and dimension tables which provides the user with the necessary information for decision making.
- A duplicate of transactional data that is specially structured for the purpose of analysis and querying.
- Beforehand, the questions that need answering needs to be known, and will exactly determine the structure of the DW, and what it can only ever answer.
- Not suited for agility in terms of delivery of value or accomodating changes in the source business.
- Due to data wrangling, a semantic gap develops between the source data and the facts and dimensions of the warehouse, requireing agressive documentation efforrts to keep under control.
- Was birthed in an ear where storage and compute was limited, and when changes in business rules were infrequent.
- For many years, Data warehouse architecture consisted of Inmon or Kimball methodology.
- Each methodology design (Star Schema vs. Snowflake Schema) has its own pros and cons but are unable to meet the requirements of handling large volume of data processing and re-engineering of data.
- Both Star Schema and Snowflake Schema are time variant, non-volatile, costly and not user-friendly.
- With large amount of data from multiple sources and regular business rules changes, Inmon and Kimball data modelling approaches become less effective.
- A better evolved model of data vault is created by Dan linstedt.
- both Star and Snowflake schema designs are optimised for query read performance of OLAP-type queries.

Data Vault

- A detail-oriented, historical tracking and uniquely linked set of normalized tables that support one or more functional areas of business, organised in a type of scale-free network.
- Inspired by presumed scale-free networks found in nature and explicityly constructed, such as neural networks, airline networks, social networks, computer networks.
- It is a hybrid approach encompassing the best of breed between 3rd normal form (3NF) and star schema.
- The design makes the model efficient to store large volumes of data and changes of business rules do not require changes in the data warehouse

hence it is cost efficient and user friendly.
- Source tracking is an integral part of the design: every record is accompanied by the name of the source file, table or other identifier.
- Design allows for accomodating varying frequencies of data loads: one entity can at the same time have both frequently, and seldomly changing attributes.
- Since all relevant data are tracked, questions the business seeks to answer need not be known in advance, in contrast with the traditional approach of crafting the data warehouse to answer specific questions only.
- Can be queried directly, or can serve as the basis from where to build out materialized information marts of denormalised views.
- Single, fixed width data type for all keys, based on hashes computed on real world business keys - zero dependence on meaningless surrogate keys from source systems.
- Single data type for all keys presents many optimisation opportunities, and simplifies joins.
- Since keys are computed rather than looked up, parallel data loads are the norm rather than the exception.
- Computed keys simplifies data loads from multiple systems into one single vault.
- Can perform change tracking through a write-only data ingest process, avoiding expensive update-based operations.
- Data errors the business systems introduced are not corrected but stored as is: data correction only happens in derived information marts so as to preserve full integrity at all times.
- Because of the modelling of business entities, their relationships between each other, and the attributes describing entities and relationships, the vault becomes an enduring model of the business itself.
  - Contrast this with a purpose-built data warehouse that is shaped by performance concerns that particular storage technologies impose and arbitrary choices in defining relations and granularity.
- All of these design patterns enable the Data Vault model to inherit scale-free attributes, meaning there is no known inherent limitation on the size of the model or the size of data that the model can represent, other than those limitations imposed by the infrastructure.

- Optimised for write at scale, with poor read performance, hence the need for data marts.

Data Vault Main Concepts

Three types of tables

Everything is modelled as one of three types of tables, with two additional tables - point-in-time (PIT) and bridge tables serving as optimisation structures:

- hub : track natural keys of entities, real-world or virtual

- satellite : track changes of hub or link properties over time
- link : track relationships between hubs over time

and the optimisation structures:

- point-in-time (PIT) : assembles together related satellite data at specific points in time
- bridge : assembles together related relationships (links) at specific points in time

In more detail:

- hub : represents real-world entities, but only their natural keys, not tracked over time, so one-to-one mapping between a real-world entity and a row in a hub
  - think of a person's ID number, a car's VIN, a PI Tag's name
- satellite : all "properties" of a real-world entity, with changes tracked over time
  - think of a tag's geo-location, engineering unit, description etc.
  - one-to-many cardinality between hub and satellite(s)
  - one record of a hub can have multiple satellites,
  - but each satellite represents similar properties of the entity, i.e. a geo_location_satellite for a tag, and a maintenance_log_satellite for the same tag
- link : relationship between hubs, tracked over time
  - a link can have satellite(s) if the relation of two or more hubs require additional context
  - good example of a link is relating a fmz_hub to a tag_hub, in fmz_tag_assignment_link
  - fmz_tag_assignment_link could have tag_assignment_details_satellite to capture which user and at what time and place this happened
  - a link could have one or more values as degenerate keys if they are required for uniqueness: example would be time of day

Other concepts

Some of the below might not make sense until the whole document is considered in full.

- Keys are always computed as hashes of the natural keys: no lookups of keys, no use of source system surrogate keys.
- All loaded records have a `_loaddatetime` column, which is not when the record "occured" but the first time the vault saw it.
- For a given load batch, the `_loaddatetime` needs to be similar for a given table: it is through this `_loaddatetime` that a load batch can be identified.
- A convention is required in terms of hashes: will they be stored as hexadecimal ASCII representations as string or as binary fields.
  - both approaches have benefits and drawbacks

  - the norm is to use hexadecimal ASCII representations to assist portablility
  - for saving space and speeding up joins, binary representations are preferred
- A convention on how to handle null values in hashes is required, with options:
  - use the string "*NULL*" or something similar
  - use the empty string
- For hashes that span multiple columns (_hashdiff and _hashkey of complex keys) a seperator char needs to be chosen, with options such as:
  - use '|' to seperate before hashing
- A suitable hashing algorithm needs to be chosen: a good choice here is sha256 with these properties:
  - very unique, very low chance of collisions
  - relatively common and fast
  - 256 bits = 64 hexadecimal ASCII chars or 32 bytes of a binary column type

How to implement

Data Vault required information

You need to have proper answers to the following questions if you plan to build a data vault:

1. Entities: identify all the real-world or virtual entities you have to model and in which source data they are found. Some entities might be spread out over multiple sources.
2. Business keys: identify the column(s) that uniquely define each entity you will be modelling.
3. Properties to track: for each entity modelled, identify the set of properties you have to track changes for.
4. Property groups and rate of change: for all properties to track changes for, group them into related subsets, taking into account rate of change such that properties changing at similar rates are grouped together.
5. Relationships and events: for all modelled entities, identify all the relationships and events involving more than one entity.
6. Relationships and events properties: for all relationships or events to model, identify all properties associated with the relationship or event that are key in defining the relationship or event.
7. Properties or degenerate dimensions: for all properties associated with relationships or events, make a distinction between properties that define the relation or event, and properties that represents additional information that need to be tracked over time.

It is important to note that you need not model your entire domain along Data Vault 2.0 before you can derive value from it. A better approach is to identify the

smallest possible subset of the answers to the above questions that would satisfy some business use case so that you can build a kind minimal viable product (MVP) data vault.

Once you have solid answers to the questions above, for a minimal viable subset of your domain , you can proceed as follows:

1. Entities: each of these will be a hub table, and they should be loaded from the various sources so that there is only ever one entity per row in the hub table - even if multiple sources define the same entity.
2. Business keys: the hashed concatenated columns will be the primary keys for the hubs.
3. Properties to track: the concatenation of the text versions of the property values will be hashed to form the hash diffs, an essential part in tracking changes.
4. Property groups and rate of change: each group of properties will form a satellite.
5. Relationships and events: each relationship or event to be modelled will be represented as a link table.
6. Relationships and events properties: some of these will be required to define the relationship or event, and some of these will be tracked as satellites off of links.
7. Properties or degenerate dimensions: all the properties that define the relationship or event will be part of the link table, and the rest will be tracked in one or more satellites off of the link tables.

Common columns to hub, link and satellite

- _hashkey : cryptographic hash such as sha2 256-bit over text of natural key
- _loaddatetime : timestamp of load, when the vault first encountered this record
  - not to be confused with the time an event occured, or measurement taken: this is simply when first the vault saw the data
- _originsource : text of file name, or database table, or api endpoint URI for data lineage from which this information was derived

Hub

- Contain base columns, and only one record per real-world or virtual entity.
- Contain one or more columns that form the natural key, to store the original representation of the natural key: usually called name or code or id.

Columns:

- _hashkey : cryptographic hash such as sha256 over text of natural key

- _loaddatetime : timestamp of load, when the vault first encountered this record
- _originsource : text of file name, or database table, or api endpoint URI for data lineage from which this information was derived
- name columns : the columns from which the _hashkey is computed
- primary key (_hashkey)

It is important to note that: 1. because the _hashkey is computed via a known algorithm such as sha256, there is never the required step to "lookup the id" while ingesting data from other sources: - one simply compute the hash. 2. no matter how many source systems feed in, each entity will have a singular identity and single row in it's hub

Satellite

Satellites persist the associated data for a hub.

The columns are:

- _hashkey : cryptographic hash such as sha256 over text of natural key from related hub
  - this is the foreign key to the hub for which this satellite tracks values
- _hashdiff : cryptographic hash such as sha256 over text concatenation of all of the property columns tracked here
  - _hashdiff makes it easy to check if a set of properties changed or not: a single comparison is required
- _loaddatetime : timestamp of load, when the vault first encountered this record
- _originsource : text of file name, or database table, or api endpoint URI for data lineage from which this information was derived
- property_1, ..., property_n : but of names that makes sense in the context
  - this is the properties that are tracked and persisted
  - recent trends additionally use JSON columns here to accomodate future yet unknown schema changes
    - See here for an example of how to use JSON in the VARIANT column type of Snowflake to achieve this.
- primary key (_hashkey, _hashdiff)

It is important to note that: 1. if a hub receives an additional body of properties to be persisted, this does not require a schema change but simply the addition of a new satellite for this purpose 2. if, for a particular _hashkey, if any of the associated properties changed, a new record is added to the satellite 3. no update to past records ever occur - this is a fundamental feature of data vault: ingest only, write only, immutability

Link

Here we track the relation between different hubs:

- _hashkey : cryptographic hash such as sha256 over text concatenation of all of the hub_1_hashkey, . . . , hub_n_hashkey values, as well as any degenerate keys such as datetime
  - this _hashkey would be pointed at by a related satellite (if required)
  - such a related satellite would capture additional properties of the relation that are not part of the degenerate keys (if any)
- hub_1_hashkey, .., hub_n_hashkey : cryptographic hash such as sha256 over text of natural keys of all hubs involved in this relation
  - this is what relates the hubs together
- _loaddatetime : timestamp of load, when the vault first encountered this record
- _originsource : text of file name, or database table, or api endpoint URI for data lineage from which this information was derived
- primary key (_hashkey)

Benefits

Adopting an approach such as this, inspired by Data Vault 2.0 includes these benefits (among other things):

- Because all keys are computed from the hashes of natural keys rather than looked up, the dependance of having to query a lookup table is removed, and ingest can happen in parallel.
- Because of the write-only architecture (compare this to merge or upsert semantics) ingest can happen faster and on a larger scale.
- Because no actual transformation of data happens, the data can be considered as raw and representing the real-world, and can power many future transforms with confidence.
  - Consider the scenario where an assumption about a scaling was proven wrong: no problem, since we still have the raw data but in a queryable form.
- Because no transforms happen, there are no confusion over at what stage something might have changed.
  - We know it did not happen in the vault layer, so it must be in a data mart built form the vault, or some data science or analytics artifact.
- Because a common procedure is followed for modelling data there is less confusion in the team over how data was modelled.
- Because changes to data are tracked it is possible to see a view of the raw data at a given point in time.
- Because the _originsource column tracks the origin of each and every record, it is easy to identify the source of data.

Data Vault in the context of Data Mesh

Having explained a little what Data Vault is, we can now turn to how it can potentially empower a Data Mesh architecture.

- Nodes could have their own vaults, with output ports being views off of their vault data.
- Because of the standardised hash for keys, nodes know how to join over output views of other nodes.
- With a common big data cloud analytics storage substrate such as Snowflake or Big Query, many nodes can build their own vaults while still being able to query others.

Examples

Tags vault example The tags vault example illustrates the simplest possible vault that can be built off of the timeseries data for tags, consisting of the tag name, measurement timestamp and measurement value.

It consists of a hub for recording tag names, a link for recording that measurement events took place and a satellite off of the link for recording the actual measurement values. It also illustrates how hashes can be pre-computed in staging tables.

Features of this vault include: 1. unique and standardised keys for tag names in the tag_hub. 1. Data lineage in the `_originsource` columns of all vault objects, making it possible to know from where data originated. 1. All load operations are idempotant. 1. Once the hashes have been computed, all loads can happen in parallel.

Next we will illustrate how to obtain useful views from the vault data: 1. Most recent measurements for all tags. 1. Point in time measurements for all tags. 1. Aggregated measurements for all tags, for a couple of aggregation windows (hourly, 6-hourly, daily, weekly).

Hierarchy organisation vault example The hierarchy organisation vault example illustrates how to track relationships and changes in the hierarchy organisation over time. It consists of hubs for each of the hierarchy levels, and a link for tracking how the levels relate to each other over time.

References

- Building a Scalable Data Warehouse with Data Vault 2.0: Linstedt, Daniel, Olschimke, Michael: 9780128025109
- Data Modeling for Azure Data Services - Peter ter Braake