Datavault

Using dbtvault in your Data Vault project

dbtvault

CONTENTS

1. INTRODUCTION

1.1. Purpose

dbtvault uses metadata to generate and execute the ETL needed to run a Data Vault 2.0 Data Warehouse on a Snowflake database.

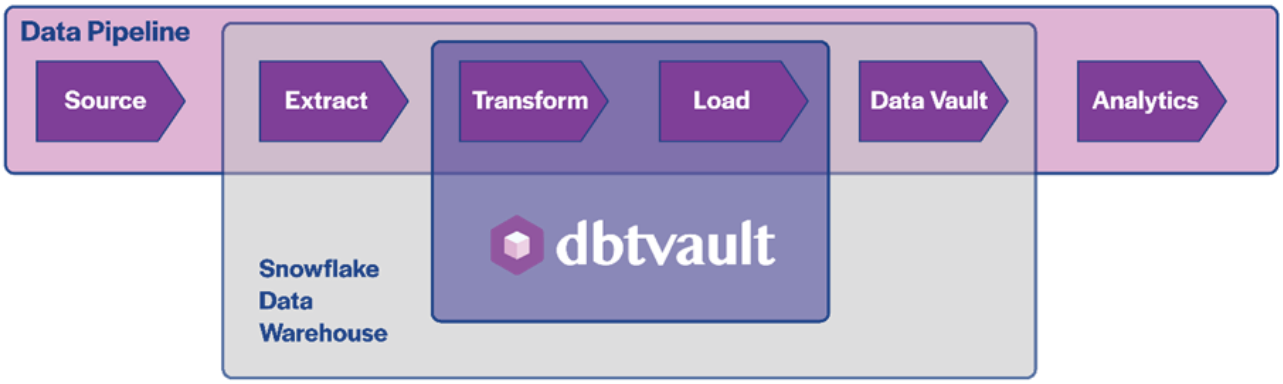
This document explains, at a high level, how to use dbtvault in the context of a Data Vault project.

1.2. Background

Data Vault is a method for the design and implementation of an enterprise data warehouse.

Enterprise warehouses are more complex than smaller point reporting solutions and need to be carefully organised and constructed to make sure they are scalable, secure and correct.

The Data Vault method delivers all of this, while making sure the development process follows Agile principles. The dbt data engineering tool and the dbtvault package make it even easier to work in a productive, Agile way.



The dbtvault package's purpose is to generate ETL that prepares a staging layer for load into a Data Vault and loads the raw data vault itself.

Our early releases have focused on the Snowflake database, and cover hashing of the staging layer and the load of the most important table types in the Raw Vault. New functionality and features will be added over time to support more database technologies and to extend the coverage to more table types and transformations.

If you have specific requirements contact us (see last page).

1.3. Where to find more information

Further information is available from the following sources:

- You need to be running dbt to use the package which is available on GitHub.
- The dbtvault package is open source and available on GitHub.
- More details, including code examples and more detailed documentation for all steps are available on readthedocs.

NOTE:

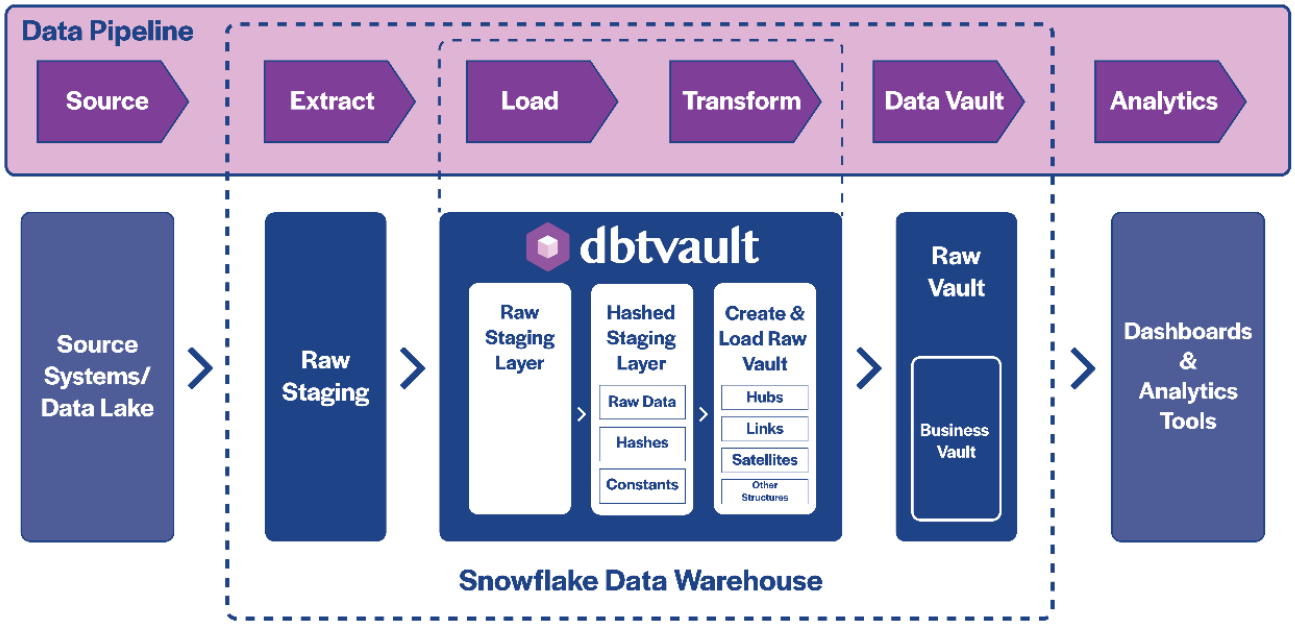
Receive notification of new features, releases and information before anyone else by leaving your email here.

2. THE DATA VAULT DATA PIPELINE

A Data Vault data pipeline brings data from source systems into a Data Vault architected database structure and then out into presentation marts: subsets of data from the Data Vault selected to meet the reporting needs of specific sets of users.

The pipeline consists of several steps, moving data from layer to layer and with each step performing a defined transformation.

The diagram below illustrates these steps and how dbtvault maps onto a Data Vault project.



Each layer of the data pipeline is described below.

2.1. Source Systems

Not a layer, this is the original source system.

dbt can query these systems, extracting data and bringing it across into a staging layer (outside the scope of our dbtvault package).

Ideally, only new or changed data since the last run should be extracted. If you can, use CDC (Change Data Capture), techniques or settings to make this easier, however, there are many ways to get data out of source systems: CSV (or JSON, XML) file extracts, querying a replica or backup copy of the live system database, streaming, activating CDC functionality, inspecting redo or replay logs, or running full table extracts and comparisons.

2.1.1. Data Lake or Persistent Staging Layer (optional)

The Persistent Staging Layer (PSL) is not part of the Data Vault standard architecture but is a common feature.

This is a kind of Data Lake, holding original copies of all the raw data fed into the Data Vault system (and may hold more than that: any data we want to retain, a superset of the data loaded to the Raw Vault).

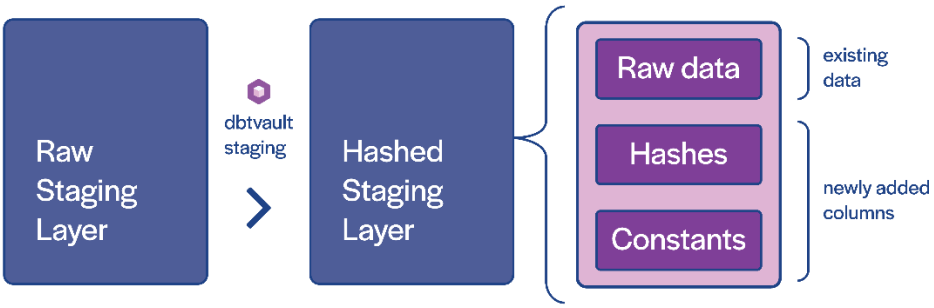
A PSL offers many advantages:

- it can provide data to replay the data feed history,
- it can hold a history of data until the Data Vault is ready to process it,
- it acts as a backup in case of problems,
- and it can expose raw tables of data as an operational data store.

NOTE:

Load of data to the PSL is not covered by the dbtvault package, however, the base dbt tool can support PSL use cases.

2.2. Raw Staging Layer



This layer contains raw data from the source systems awaiting processing. It is held in a database that the Raw Vault ETL can access directly (usually in the same database or one that can accept a cross-database SQL query).

Staging layer tables are truncated and loaded with a fresh data feed every load cycle.

NOTE:

Load of the Staging Layer is not covered by the dbtvault package, however, the base dbt tool can support staging load use cases.

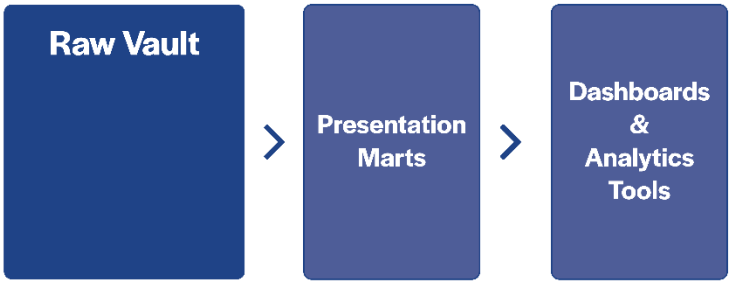
2.3. Hashed Staging Layer

The staged data needs extra columns added to it to prepare it for load into the Raw Vault. These include hashes of primary keys, hashdiffs, and implied columns from the context of the data set, such as source system codes.

NOTE:

The dbtvault package supports this layer's processing for the Snowflake database.

2.4. Raw Vault Layer



Your data modeller/architect will have designed a Raw Vault data model made up of standard components, the Hub, Link and Satellite structures used by the Data Vault method. They will also have mapped source data into the Data Vault structures.

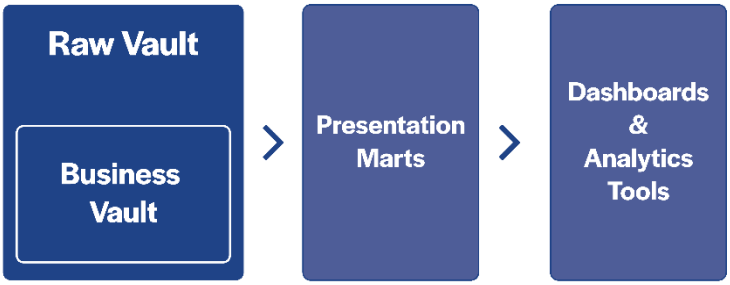
Staging tables can have dozens or hundreds of columns containing information about multiple data concepts and will map across to and populate several Hub, Link and Satellite tables.

As we have this mapping detail we can write a set of dbt modules, one per target table, containing the mapping metadata and invoking macros from the dbtvault package. These models will generate and run the SQL needed to populate the Raw Vault.

NOTE:

The dbtvault package supports this layer's processing for the Snowflake DB.

2.5. Business Vault



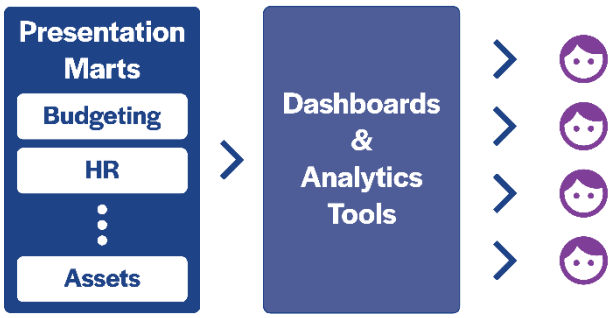
The purpose of the Raw Vault is to split up source data, irrespective of the source system, and to sort it and integrate it across key business entities (i.e. hubs, and business structures and events modelled by links).

The Business Vault is not a layer as such, it processes the Raw Vault data and adds new, derived information as an overlay on the hubs and links constructed in the Raw Vault. Derivations can be simple, i.e. ratios of two columns, or complex, i.e. using raw data in a deep learning model to generate and store new information.

NOTE:

Load of the Business Vault is not covered by the dbtvault package, however, the wider dbt tool can support many of the business rule calculation use cases.

2.6. Presentation Mart Layer



This layer is, generally, made up of several Presentation Marts.

The Data Vault contains ‘all of the data’, i.e. a comprehensive set of raw, granular data covering the full scope of feeder systems.

Users, or groups of similar users, work with subsets of this data for their own day-to-day analytic and reporting needs. For example, finance users need finance data, HR accesses HR data, and each group will see data specific for their role, or may

even not have a legitimate need to see data their role has no need for (much of the HR data would be classified as personal or personal sensitive information).

This layer extracts several subsets of the Raw Vault data, organises them for reporting purposes, and exposes each to a selected authorised set of users. This layer might build star schemas, expose a big table of data, or construct an API.

NOTE:

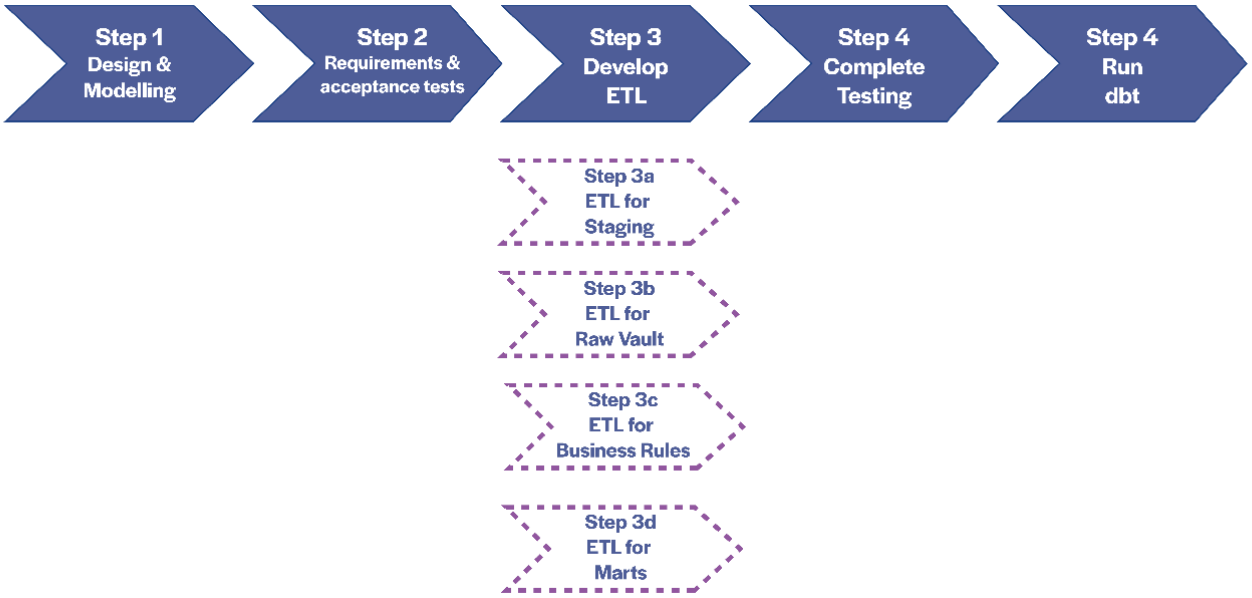
Load of the Presentation Marts Layer is not covered by the dbtvault package, however, the wider dbt tool can support many of the Presentation Mart load use cases.

3. THE WORK OF A PIPELINE DEVELOPER / DATA ENGINEER

So how does dbt work?

How can we use it as part of the development process?

In this section we give a brief outline, step by step.



3.1. Step 1 – Design and Modelling

Your data modeller/architect will have done some work before you get involved. They will have identified source systems, worked out the mechanism to get data from these systems, designed the Data Vault model, and produced a preliminary map of source data to the Data Vault. They will issue requirements – these will include building pipeline steps, and mapping data through the pipeline.

3.2. Step 2 – Requirements and acceptance tests

You'll want to clarify the requirements and write some acceptance tests.

3.3. Step 3 – Develop ETL

We won't cover the load of the staging layer or Persistent Staging Layer load, as these are outside the scope of the dbtvault package – but please don't think we are trivialising those steps.

3.4. Step 3a: Develop ETL to extend the staging tables

Write a dbt model to extend each raw staging layer table, add columns as needed, and perform any pre-processing, such as type conversions, padding or masking.

Using the multi_hash macro, specify each new column you want to add as the result of a hash.

You'll need to add a primary key hash for each hub and link table which the staging table populates.

List the columns involved (usually one for hubs, but some hubs and all links can have compound primary keys), and supply the new column name (we usually name the column after the target table and add a post-fix of '_pk', e.g. a hub_customer would have a pk column named 'customer_pk', but it's up to you).

Make sure you list columns for the same target in consistent order across staging tables, so the hash generates the same output.

You'll need to add a hashdiff for each Satellite.

Use the multi_hash macro and list all the Satellite payload columns as the first component of a triple. Provide a name/alias as the second argument to the triple to use for the staging table for the hashdiff column (we usually use the name of the target Satellite and add '_hashdiff' to the end e.g. for the 'sat_customer_details' table we would use 'customer_details_hashdiff').

Columns should be sorted in alphabetical order; If you provide 'true' as the third argument in the hashdiff triple, the macro will sort the columns alphabetically for you anyway.

Using the add_columns macro, you can provide a dbt source to automatically bring in every column from the raw staging table. Optionally, you may add aliases for columns you want to represent with different names in some vault tables. For links, for example, you may want to add the post-fix of '_fk' instead of '_pk', to denote a foreign key.

Make sure the staging table has columns holding details of the source (a code used to identify the source system and table, you manage this code list, it can be a simple integer lookup or some kind of name abbreviation), the load datetime and business effective date. If these aren't present, we need to add these as extra columns using the 'add_columns' macro.

Finally, make sure the staging table contains data for one load datetime only, i.e. for one batch only. If it holds a mix of dates

you can write a view on the table to filter out the date you want to process and run the vault load against the view.

You will also need to add the 'from' macro call to the bottom of your model and provide your dbt source as its argument.

Your dbt model should be named <stage table>_hashed.sql (or whatever you adopt as your local standard). dbt works by creating target tables or views based on the name of the file it executes. Irrespective of whether we materialise the stage hashed table or not, the model will make sure the '_hashed' layer is created.

3.5. Step 3b – ETL for the Raw Vault

Now we need to define the Raw Vault load.

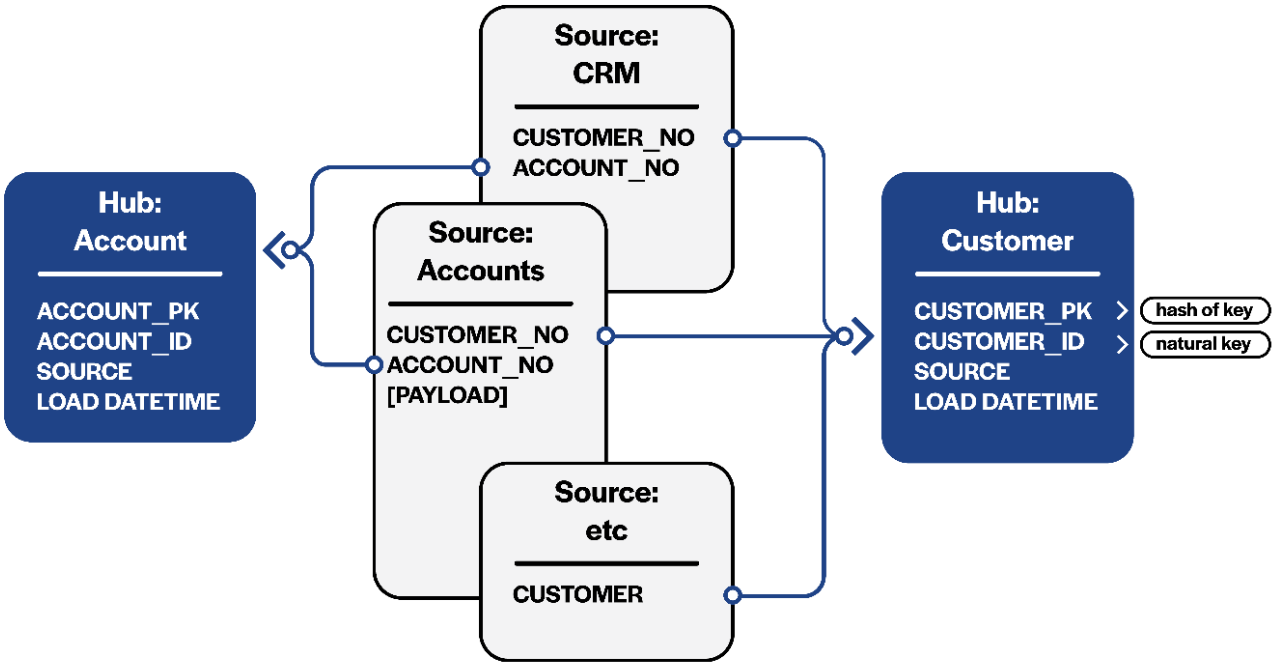
We create one model for each hub, link and Satellite in the Raw Vault model. Let's look at each in turn.

3.5.1. Hubs

Hubs are simple structures and they store a list of the primary keys of entities of significance to the business (think sku, employee, customer, site, etc.).

They load:

- a hashed primary key (a hash of the natural key, prepared in the stage hashing step)
- the natural key
- the load datetime
- a source-code



We provide this mapping data inside the model – we identify the source of the hub data (the hashed staging table we feed from), the hub column names, the stage table columns that map to these columns, and we invoke the SQL generator macro.

We name the Hub model after the name of the Hub table, and we select incremental materialisation - so dbt creates the Hub table the first time it encounters it and adds data incrementally to the table for each load afterward.

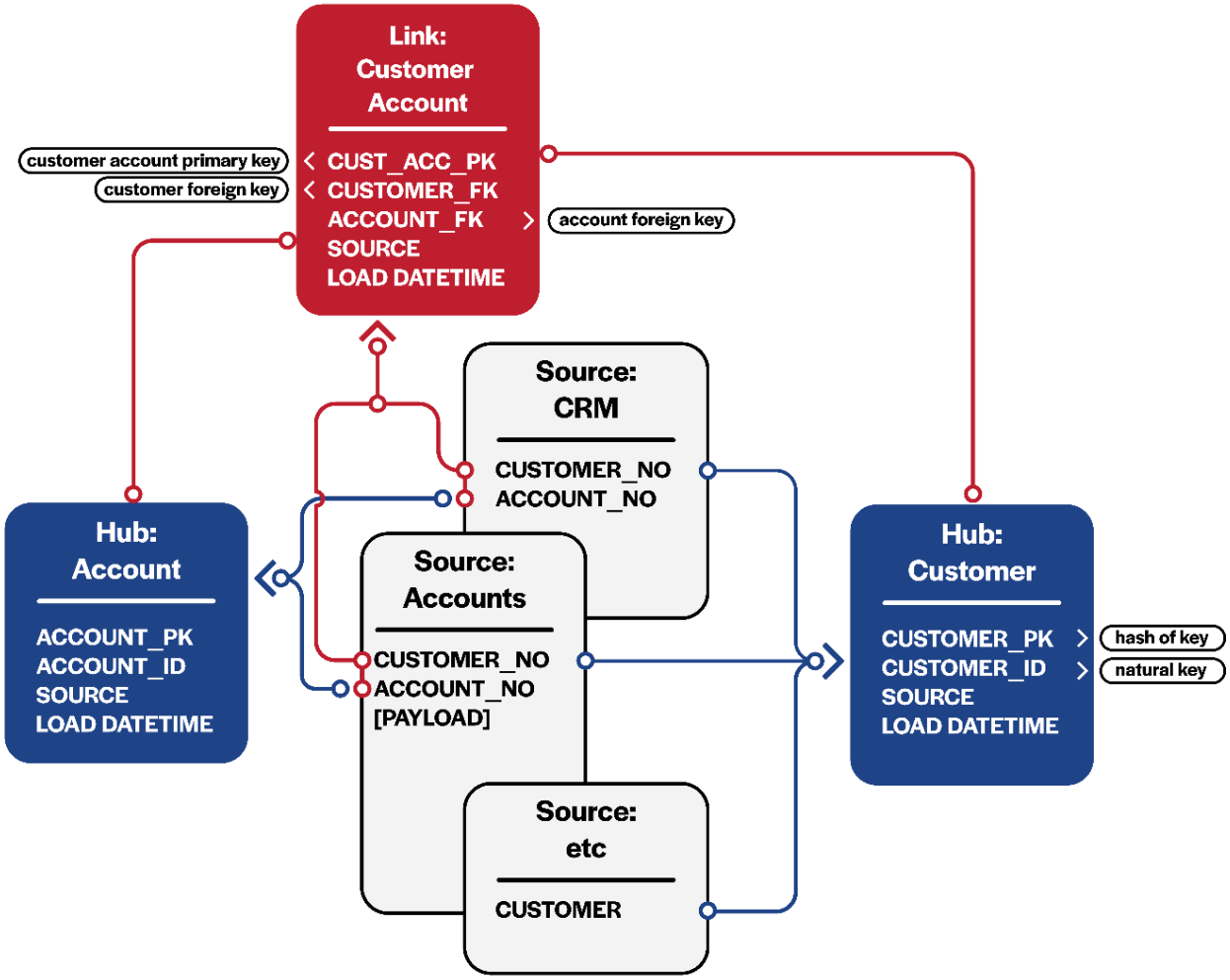
There is a quirk in the way dbt works. In Data Vault systems, multiple staging tables may map into a hub. This means multiple loads need to happen, one from each staging table. However, dbt restricts us to one model per target table. We work around this as follows: say a hub loads from four separate staging tables, we first identify the columns from each table that map into the hub, and then UNION these into a single table before we load the hub as a single operation.

The macro can cope with a union feed of mapped columns from multiple stage tables. We first need to ensure that our columns are named the same in each stage table. Use the add_columns macro in the previous step. Next, we provide references to each stage table as a list and provide source and target data as normal; the Hub model then generates all the SQL necessary to perform the union and load the Hub table.

3.5.2. Links

Links are also simple structures. They bring in:

- a primary key (a hash of the natural keys of the dependent hubs)
- the primary key of all hubs joined to the link (i.e. foreign keys),
- a source
- a load datetime value.



We write a model file just like we did for the hub, naming it after the link table.

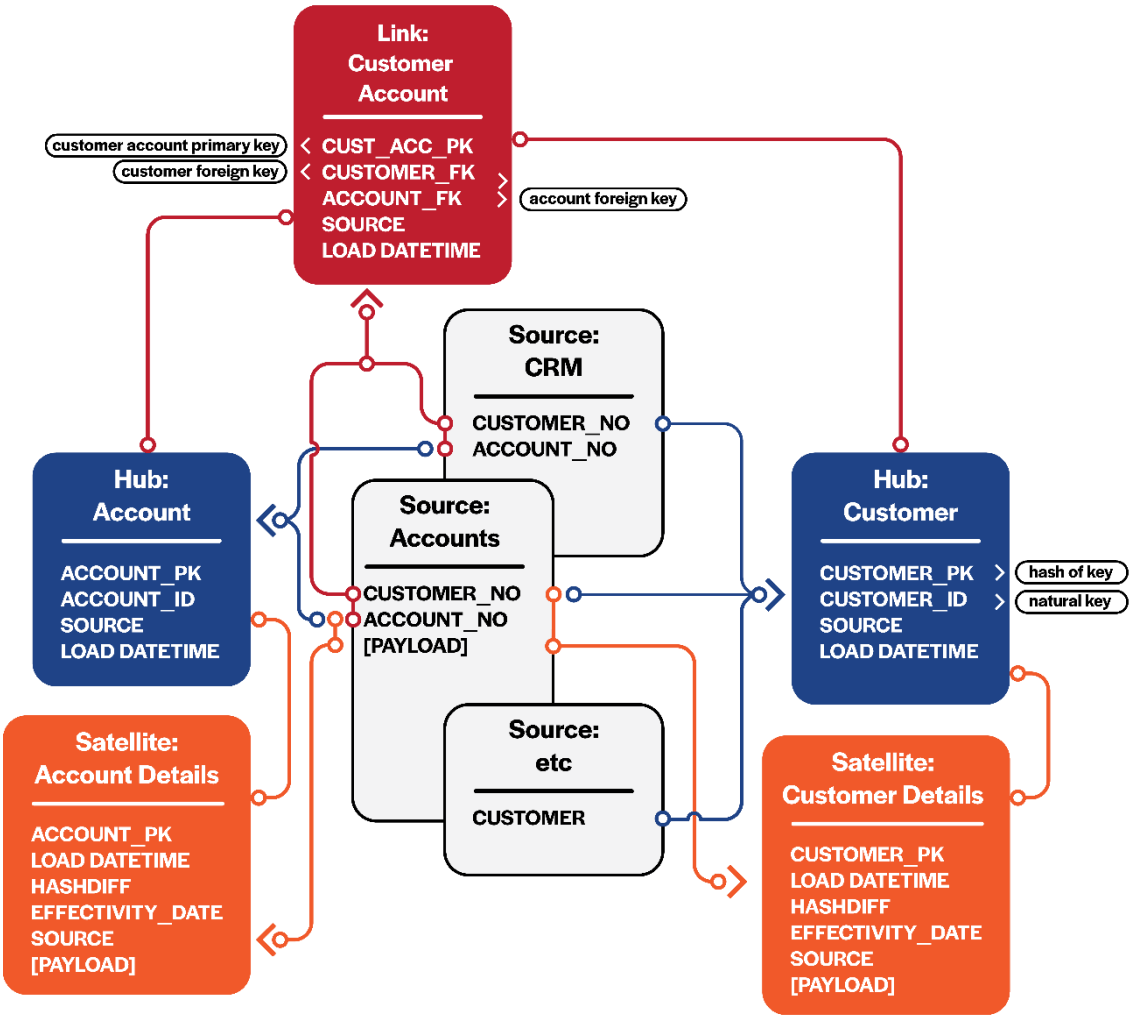
Links have the same behaviour as Hubs when they are fed by more than one staging table: we provide the stage table references as a list and ensure all columns are present in each stage table, and the macro takes care of the union and generation of SQL to perform the load.

3.5.3. Satellites

Finally, we need to define a model for each Satellite. Satellites are a little more complex, but not overly so.

Satellites have more metadata than hubs or links and a payload of one or more columns. The metadata includes:

- the primary key of the parent hub or link
- the load_datetime
- a source identifier
- an effective_from datetime
- hashdiffs
- a list of payload columns



The first few columns of metadata are the same as for the hubs and links.

The effective_from data is a business date, i.e. the date and time for which the payload has significance for the business, and this may be different from the load_datetime, especially if there is a delay incurred by batching the feed.

The hashdiff is the hash of a string made up of the concatenation of the primary key and all payload columns, sorted alphabetically. The hashdiff is used to find out if the payload has changed in each feed, so by comparing hashdiffs of the latest loaded Satellite we can work out if there have been any changes and decide if we need to load new Satellite records.

3.6. Step 3c – ETL for Business Rules

Business rules add data derived from raw data items.

This derived data can be the ratio of two columns, the results of data cleaning, aggregation, or a more complex calculation.

The dbtvault package does not support the business rules calculations. However, these are supported by the vanilla dbt application.

Simply write a model, include SQL in the model to calculate the derived values. Name the model after the Business Vault table (i.e. a Business Vault Satellite) you want to create and dbt will handle the rest.

3.7. Step 3d – ETL for Marts

Finally, define the load of your marts.

Marts can be big reporting tables, star schemas, or other reporting friendly structures.

Data Vault has techniques that create Dimension or Fact tables as views on the Raw Vault. Check these out in the Data Vault book (“Building a Scalable Data Warehouse with Data Vault 2.0” by Dan Linstedt and Michael Olschimke).

The vanilla dbt application supports the generation of SQL to load marts.

Simply write a model, include SQL in the model to populate the target tables, dimensions and facts. Name the model after the mart table (i.e. a Dimension or Fact table) you want to create and dbt will handle the rest.

3.8. Step 4 – Complete testing

As each model is defined, run your functional tests to show that the mapping and transformation models work as intended.

Check completed work in regularly to version control.

3.9. Step 5 – Run dbt

You can run dbt from your command line, or for production purposes, you can run dbt on a VM in your data warehouse environment.

There is also an offering included with dbt itself called dbt cloud, which provides scheduling, logging, documentation integration and many other features which aid with continuous integration and running dbt in production.

dbt will:

- compile your models, wrapping and generating SQL from the metadata data provided
- work out dependencies between models
- run the SQL, respecting the dependencies
- run parallel loads, up to the number of streams you specified in the configuration file.

ABOUT US

Datavault

Datavault is a UK based consultancy specialising in helping our clients maximise their investment in data warehousing and Business Intelligence and analytics.

We work with clients throughout the life cycle of projects from strategic alignment and business case development, through to design, implementation and ongoing operational support.

Why not talk to us about your project? We supply a range of services around the following themes:

- Helping clients modernising their Data Analytics service
- Adopting Data Vault 2.0 with coaching and training
- Enabling clients to get the most of their data with Information Governance
- Migrating your Data Warehouse to the cloud

Whilst we are based in the UK, we have worked with clients in other countries.

dbtvault

dbtvault was developed as a result of our experience helping clients get started quickly and efficiently with their Data Vault projects, such as developing fast prototypes to prove the efficacy of the Data Vault method.

Contact Us

Please do get in touch, we are always interested in hearing about people's experiences with dbtvault.

email: enquiries@data-vault.co.uk

tel: +44 (0) 23 92 63 7171

web: www.data-vault.com

twitter: @Datavault_UK