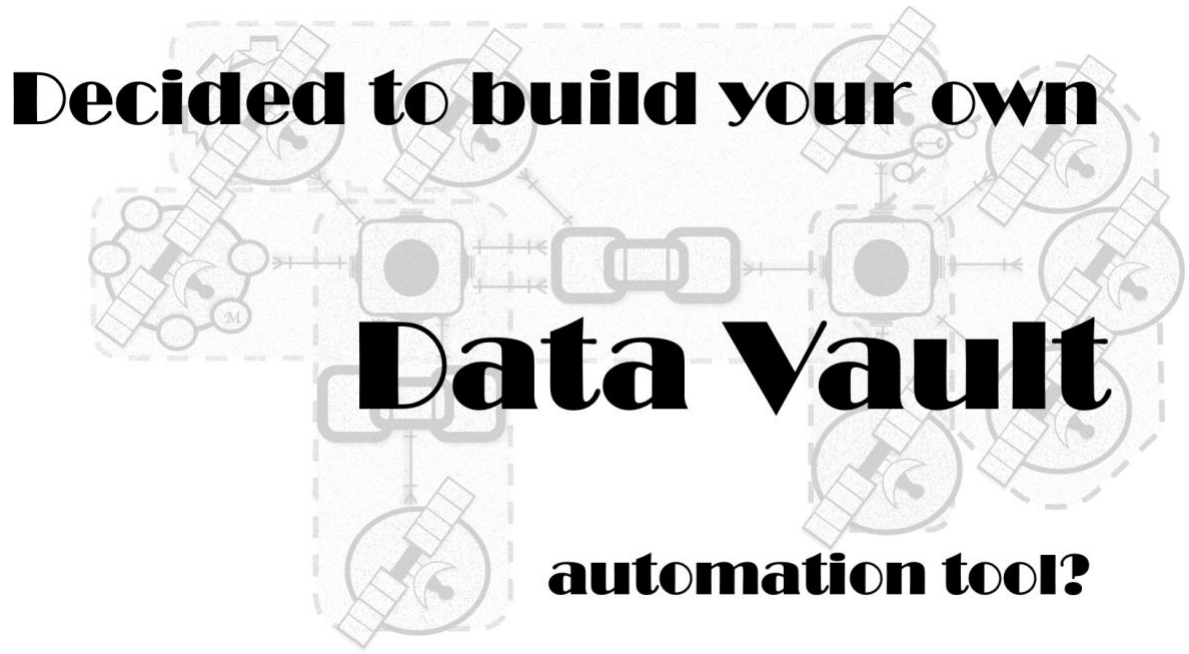


Decided to build your own Data Vault automation tool?

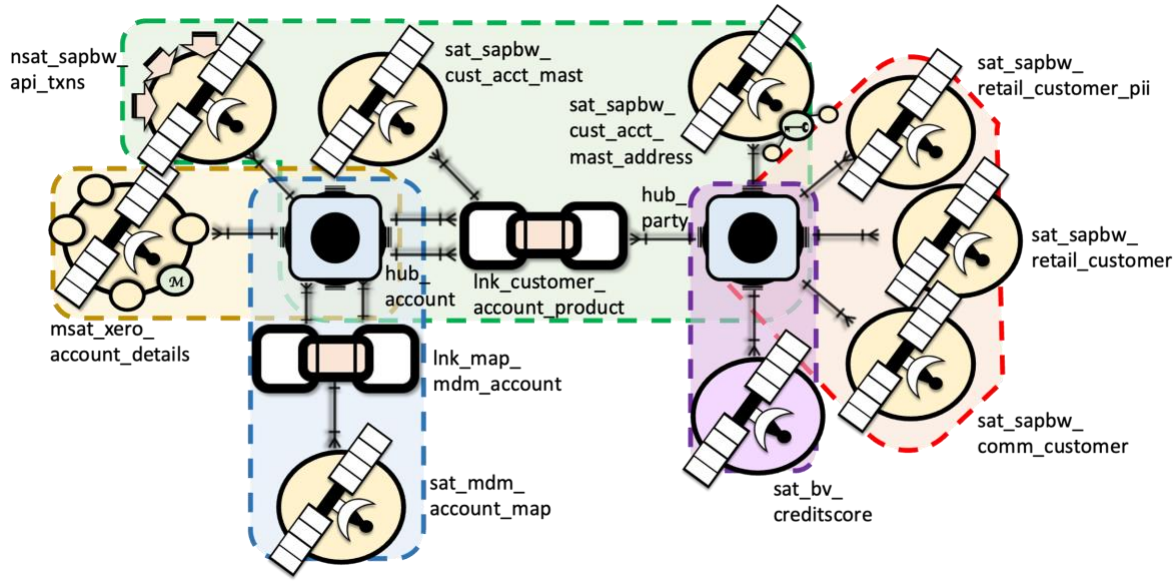
With a focus on data **mapping** and **automation**, customers building *their own* Data Vault automation tool often resort to an Excel frontend, in this presentation we will define such a structure that is easily **configurable**, intuitive (once you get used to it), and above all, **scalable**.

Yes, this does follow Data Vault 2.0 standards and include a few additional ***pragmatic*** configurations that maybe even the commercial tools are not able to deliver. But that's just one of the reasons you may pursue building their own tool, being able to rapidly deploy the functionality you need without waiting for a fix or release from a commercial vendor. This is not to discredit the commercial automation tools! What you get with those is the patterns already configurable and supported by their experts, the downside is you need to wait for new features and updates to happen in most cases. Ultimately the choice is yours! *Build or Buy!*

In this paper we will,

- Introduce a *fake and generic* data model with as many data vault modelling **patterns** as possible.
- Introduce the Excel front end and why it is configured in that way.
- Incrementally introduce the model to the mapping document and build from a simple mapping to a more complex one. Note that of course if you do not need all the advanced techniques in this template then by all means skip it!

Data Vault model



Some creative license applied to the naming and content but included to illustrate concepts within DV2.0, in reality only some of the nuances modelled here you may *never* encounter depending on the source system and business being modelled.

To explain the data model, start by the hub tables involved, *always*

1. **Hubs** – there are two of them

- **hub_party** – often higher-level data models will describe **business objects** as parties, parties to a transaction, parties to a contract *etc*... This generic business object acts like a super-type of *organization* and *person* as both could be interested parties but are distinctly different business objects (and have their own unique non-clashing ***business keys***). We will adopt the party representation because our source is modelled that way. Do not build a **party model** if that is **not** how your business is being represented! Doing so adds a point of maintenance within the data platform and doing so will forever require manual intervention to the model. *Not good!* If you pursue a party model let the business be responsible for what defines a party, *not you!*
- **hub_account** – a ***common*** (shared) hub we see in multiple industry verticals is the humble account hub. Be careful in the naming of these because an account can mean so many different things depending on the business! A hub is modelled based on semantics and grain and therefore getting this right early is **pivotal** to a **scalable** data model. For example, is an account in Salesforce the same as an account in SAP? *Likely not*, do we have a **composite key** representing a bank account and a **simple key** representing an account id for a credit card account? *Likely so*. Remember, concatenating business keys into a single hub table column is ***not* best practice**. Concatenating keys into a single column means every query utilizing the hub table must derive the original key from the concatenation (and selectively), an unnecessary querying burden if you could have already solved this integration debt upfront. Remember that when we explore the model, solve the integration upfront rather than leaving it to the analysts and reports to solve this at query time! *A composite key indicates a different grain to a simple key, keep that in mind*.

2. **Links** – there are two of them

- **lnk_customer_account_product** – only two link tables are represented in this model, a link is a ***many-to-many table structure*** that can house **any cardinality** of relationships. This makes it ***scale***, it also means you must understand that cardinality

especially in resolving querying data from the data model. This modelled link table has two cardinality lines connected to the hub_account table and only one to line to the hub_party table, this is because we have two identifying account ids for a single account to one party (the second account id is optional – think zero-key concept). *Do you categorize this as a same-as link then?* But then why did I include the party? There's good reason for both questions. First, yes there is a same-as relationship but think of it as a **logical representation** when modelling it into a data vault because how you query a same-as link as opposed to a regular link table is **identical**. Secondly, in data vault we need to ensure we can accurately **rebuild** the source if we need to, a form of **model-to-source audit trail**. If we break up the unit of work, it might become far more difficult to recreate that said source.

- **lnk_map_mdm_account** – an integration representation that the MDM source sends over. The MDM id is subject to match-merging rules. *More on this integration later!*

Noticed the multiple colours in the data model? Yes of course you did. We have different sources and business processes contributing to a single data vault model. Hubs integrate those business models and links represent the unit of work between those participating business objects. The ***raw vault satellites*** are recommended to be ***source-specific*** whereas ***business vault satellites*** derive additional value to an existing business process or consolidate additional analytics from various raw and/or business vault content.

3. **Satellites**

Green – source sapbw

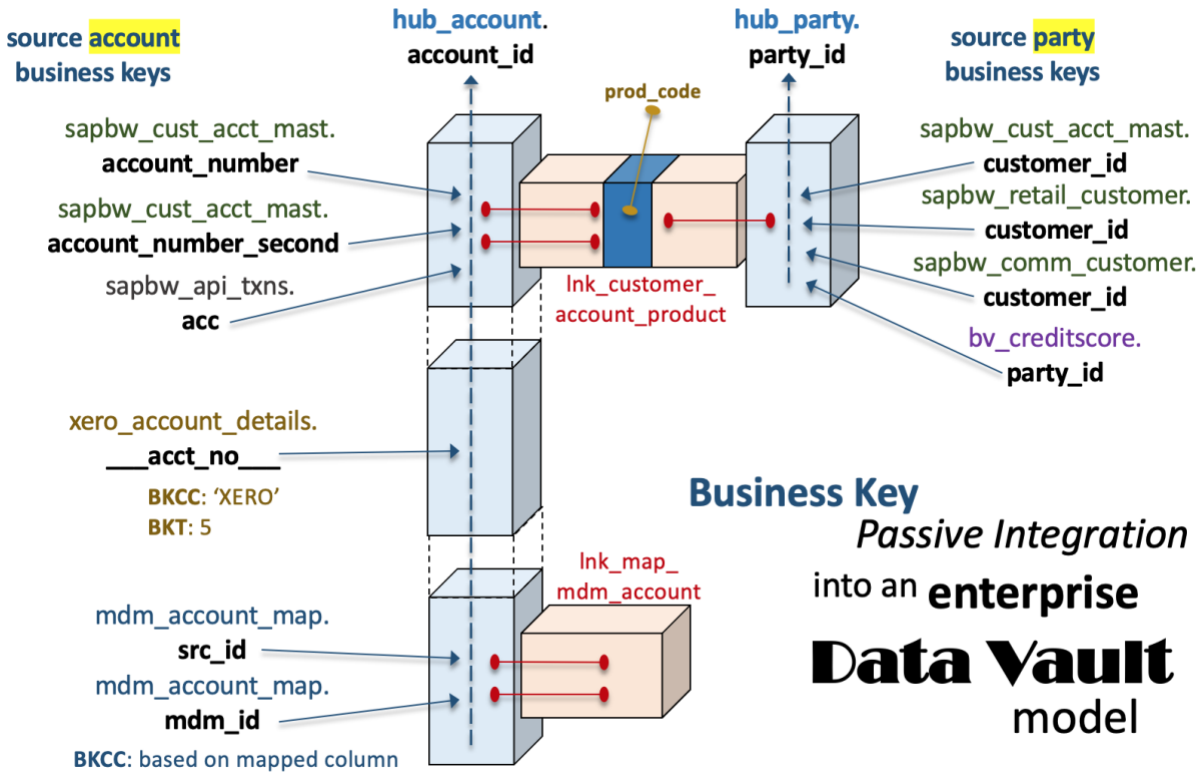
- **sat_sapbw_cust_acct_mast** – a link-satellite with descriptive details about the unit of work it is a child of.
- **sat_sapbw_cust_acct_mast_address** – an offspring of satellite splitting, the content here repeats across multiple instances of the unit-of-work and thus querying this content if it were left to the link-satellite would **require deduplication**. An unnecessary burden to reports and analysts and thus we solve this upfront by **splitting** the content to the grain of what it represents, party-addresses. Also note that this satellite has a ***dependent-child key*** called address-type. A party might have one or more active addresses by type and thus we track changes ***to that*** address-type
- **nsat_sapbw_api_txns** – ***non-historised satellite***, this representation is needed if the content coming from source is ***immutable***, always by definition "a change" and would not benefit from checking if the new record from the source is actually new (the data vault model has not seen this record before). Non-historised content is ***transactional and near-realtime***, if transactions were being supplied in a ***batch***, then this is not the satellite to use, you could be using a regular satellite or a satellite with a dependent-child key where the dep-key is identified as the ***intra-day key***.

Red – source sapbw

- **sat_sapbw_retail_customer** – retail customer details are stored here and integrate by the ***same business key***; the single source uses a ***single immutable business key value to represent the same customer***.
- **sat_sapbw_retail_customer_pii** – another offspring to satellite splitting but with a slightly different purpose. The common property of a PII content is that it is **identifying**, in other words this content will hardly (if ever) change. This means that this satellite might record only a single descriptive row for a party and makes it *far easier* to manage identifying content in isolation rather than having to manage such content if we were still dealing with changes to the non-pii data if they were integrated into one satellite

- **sat_sapbw_comm_customer** – to demonstrate that party can be either retail or customer our model will load both sets of keys into a **single hub**. They will never *collide* because they have their own independent keys and what this means is that if you were to execute an equijoin between the hub and the above three satellites (retail and commercial) you will get **zero records returned**. Intentionally, because you should not return records in the commercial satellite if you are querying about retail customers.
  Blue – source mdm
- **sat_mdm_account_map** – descriptive details of the matching process
  Yellow – source xero
- **msat_xero_account_details** – a multi-active satellite where a **change in SET** inserts a new **SET of records** to the target multi-active satellite table.
  Purple – source raw vault
- **sat_bv_creditscore** – illustrating the proper way to represent the derived content (business vault).

Passive integration of Business Keys in an enterprise Data Vault model



Part of the challenge of being a Data Vault practitioner is to make best efforts to passively integrate by key. Yes, this requires data profiling, but this also requires having the right people in the room. If you get this right not only will you reduce the number of tables required and have a model that scales, but ensure that the data will flow!
Peruse these articles to understand more

- Data Vault Recipes
- Building Data Vault modelling capability through the Mob

Excel Front-end

To ensure scalability of automation the Excel frontend is represented by a three-tab system. Those three tabs are:

- **src_to_target** – source to target mapping, a sort of vertical representation of the data mapping by representing source file / table columns to target data vault artefacts
- **dv_to_dv** – mapping between data vault tables, required to identify the parent-child relationship between hub and hub-satellite, hubs and link and link and link-satellite.
- **dv_lnk_eff** – ensuring that a single link table can have as many driver-key + effectivity satellites as needed.

And now for the tab content description…

src_to_target

column	description
source_column	Column in source table,
column_order	Ordinal value for consistent **HashDiff** calculation
source_table	The source file / table / API endpoint
dv_dependentchild_key	'Y' if the column is being used as a dependent-child key in a link and/or satellite
dv_business_key	Business key **conformed** to a column name representative as the business object. *Always* cast the business key to a text string.
dv_business_key_order	In the case of a composite key the ***ordering is vital*** to ensure consistent surrogate hash key generation
dv_bkeygroup	A unique arbitrary value you assign that ties composite keys together or represent the simple key for linking to **dv_to_dv.remap_source** column Used in conjunction with **dv_to_dv.remapped_hashkey** column to designate a rename to the hash key column name
dv_target_table	Hub, link or satellite table
dv_tabletype	Used for the variance in target table **load types** • link – link table • nlink – non-historised link table • sat – satellite table • msat – multi-active satellite table • nsat – non-historised satellite table
dv_tenant_id	Should the model structure be shared by **multiple tenants** an id is included in surrogate hash key generation and used as a filter clause in all data vault structures. Set to a default value if not needed
dv_jira_id	For the effort to build the model the jira id is included in all related artefacts, subject to change when tasks are assigned to apply changes to the data vault model
dv_business_key_code	Used only if a business key collision could take place, relates to the messaging around equijoins, if two business

column	description
	keys in the **same hub table** match but represent **difference business objects** then you could be joining data that does not relate to that object. *Do not make this a default value tied to a source!* A business key could be shared between source systems and thus including defaulted tie-breaker value by source will create integration debt! (more tables to join than needed)
dv_tag	When the Excel model cannot cater for **overloaded data columns**, or values need to be *carried* from raw / business vault then this column identifies a ***model-driven column*** with the values needed. The values can be • DV_BKEYCOLCODE – the source column contains a row-level assigned *business key collision code*. (requires mapping) • DV_APPLIEDDATE – column will carry the applied date into the target data vault table • DV_BKEYTREATMENT – row level assignment for non-standard business key treatments
dv_tag_group	Used in conjunction with dv_tag, a unique arbitrary value ties the business key collision code with the business key it applies to. Imagine a landed file has multiple business key columns so which business key columns should this tagged column be applied to? The flexibility means you could be assigning it to more than one business key column or just the one!
dv_business_key_treatments	Special case, used when a source may (for example) allocate business key ***case sensitivity*** as different business objects, i.e. 'A' and 'a' are not the same, that means the default business key treatments cannot apply! This is extremely rare and may instead indicate a poorly designed business key or a lack of business key constraints at the source!

dv_to_dv

column	description
parenttable	The table that is the chief place to find the surrogate hash key Hub – **hub-hash key** Link – **link-hash key**
childtable	Table that must match by surrogate hash key to its parent table • A *hub* satellite whose surrogate hash key comes from a *hub* table • A *link* satellite whose surrogate hash key comes from a *link* table • A link whose surrogate hub-hash-keys comes from one or many hub tables
source_table	The same source as defined under src_to_target, *important to delineate for hubs and links*

column	description
remapped_hashkey	The default generation of a hub hash key follows a templated form based on the hub table name itself. When representing a relationship between *two or more business objects* whose business key resides in the *same hub* at least *one* of those surrogate hash key names **cannot** follow the standard convention, the name of the column itself must be **remapped**!
remap_source	The unique arbitrary value that came from **src_to_target.dv_bkeygroup** (see above)
hub_order	To consistently produce the surrogate link hash key the order of the contributing hub-hash-key source **must** be the same! Must include the collision code and tenant id columns in the code but **not** necessary to map in the sheet!
dep_key_shared	Because identifying groups of business keys is mutually exclusive to identifying the dependent-child key the same **src_to_target.dv_bkeygroup** column can be used to identify dependent-child keys that need to be loaded to more than one satellite in a satellite split scenario
dv_assign_recordtracking	Generate a hub or link ***record tracking satellite***
dv_assign_statustracking	Generate a hub or link ***status tracking satellite***, the source table should be a **snapshot** of the source system table or equivalent.

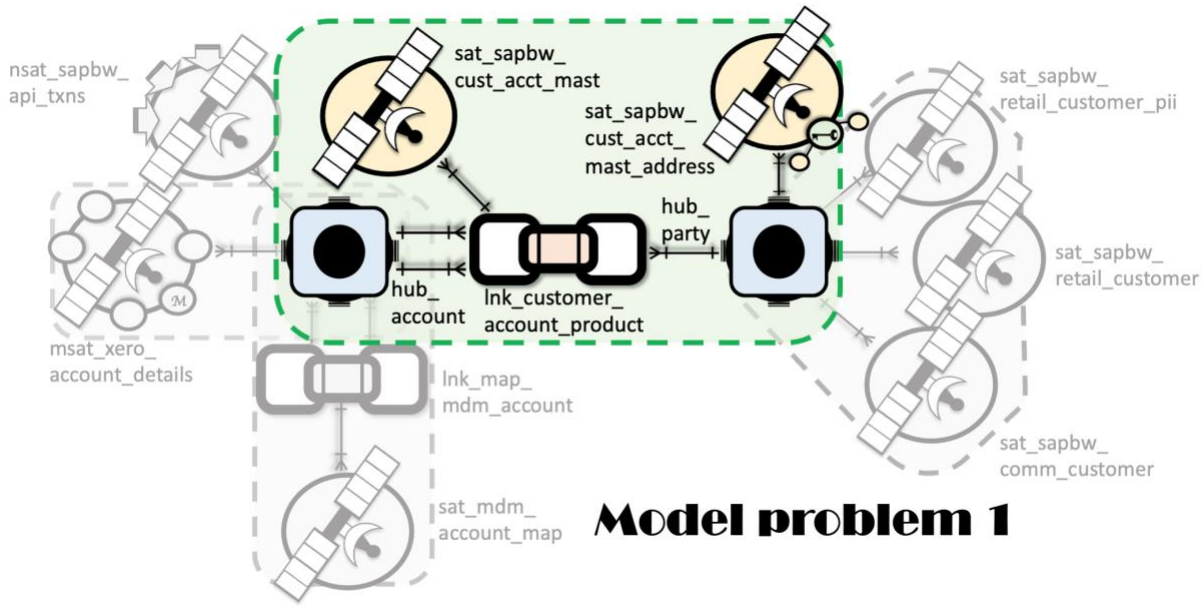
dv_lnk_eff

This will not contain a replica of what was defined in **dv_to_dv** mapping, ***effectivity satellites*** based on ***driver keys*** does not need to track the non-driver portion of the relationship, it is intuitively known what those non-driver business objects are by referring to the other tabs!

column	description
hub_table	Driver key (or keys)
lnk_table	Link table effectivity satellite will be based on
source_table	Which source to link relationship we are tracking
remapped_hashkey	In the case of tracking effectivity on a same-as relationship
eff_sat	Effectivity satellite name

Modelling problem 1: Src: sapbw_cust_acct_mast

To better understand the Excel mapping, modelling scenarios will be designed using the three-tab Excel system. Satellite splitting is often such an *underrated activity*, but it is vital for ensuring **scalability** and **longevity** of the overall data model! Raw vault columns are brought in and loaded "as is" in column naming and data type. Business keys loaded to hubs must be conformed to a common business object naming standard and they must be (or converted to) a text data type to ensure there will never be a need to refactor the model!



Yes, a single source file can lead to multiple data vault artefacts! For demonstration purposes we are defining a few data vault artefacts and we will see how to map this model!

- **Two** business objects are identified by **three** business key columns. Two of the business keys are separate representations of the *same business object* at the same grain and thus loaded to the hub_account table. The other is the customer and it loads to the hub_party table. The three business key columns represent the ***unit of work***, and the secondary account id is *optional* and thus populated using the zero-key concept when the value is null.
- The descriptive content is split by what it represents,
    - The link-satellite sat_sapbw_cust_acct_mast is a regular satellite whose parent is the link itself
    - The hub-satellite sat_sapbw_cust_acct_mast_address has a ***dependent-child key*** because address_type column is independently tracked for the customer. A change in address by address_type will cause the new staged address to supersede the current address by address-type in the satellite.
- Product_code enjoys a one-to-one relationship with an account and is represented by a code loaded as a ***degenerative dimension*** in the link table itself.

Let's see how to map this using our Excel template!

Step 1: Map source to Target tables and mapping the related artefacts

- src_to_target

source_column	column_order	source_table	dv_target_table	dv_table type
account_number	1	sapbw_cust_acct_mast	hub_account	
account_pk	2	sapbw_cust_acct_mast	sat_sapbw_cust_acct_mast	sat
account_number_second	3	sapbw_cust_acct_mast	hub_account	
customer_id	4	sapbw_cust_acct_mast	hub_party	
cust_pk	5	sapbw_cust_acct_mast	sat_sapbw_cust_acct_mast	sat
prod_code	6	sapbw_cust_acct_mast	lnk_customer_account_product	link
create_date	7	sapbw_cust_acct_mast	sat_sapbw_cust_acct_mast	sat
address_type	8	sapbw_cust_acct_mast	sat_sapbw_cust_acct_mast_address	sat
line1	9	sapbw_cust_acct_mast	sat_sapbw_cust_acct_mast_address	sat
line2	10	sapbw_cust_acct_mast	sat_sapbw_cust_acct_mast_address	sat
line3	11	sapbw_cust_acct_mast	sat_sapbw_cust_acct_mast_address	sat

same hub table

two business key columns in the file

- dv_to_dv

parenttable	childtable	source_table
hub_party	lnk_customer_account_product	sapbw_cust_acct_mast
hub_account	lnk_customer_account_product	sapbw_cust_acct_mast
hub_account	lnk_customer_account_product	sapbw_cust_acct_mast
lnk_customer_account_product	sat_sapbw_cust_acct_mast	sapbw_cust_acct_mast
hub_party	sat_sapbw_cust_acct_mast_address	sapbw_cust_acct_mast

parenttable	childtable	source_table
hub_party	lnk_customer_account_product	sapbw_cust_acct_mast
hub_account	lnk_customer_account_product	sapbw_cust_acct_mast
hub_account	lnk_customer_account_product	sapbw_cust_acct_mast
lnk_customer_account_product	sat_sapbw_cust_acct_mast	sapbw_cust_acct_mast
hub_party	sat_sapbw_cust_acct_mast_address	sapbw_cust_acct_mast

same hub table

Step 2: Identify and map the business keys

- src_to_target

source_column	column_order	source_table	dv_business_key	dv_business_key_order	dv_target_table
account_number	1	sapbw_cust_acct_mast	account_id	1	hub_account
account_pk	2	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast
account_number_second	3	sapbw_cust_acct_mast	account_id	1	hub_account
customer_id	4	sapbw_cust_acct_mast	party_id	1	hub_party
cust_pk	5	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast
prod_code	6	sapbw_cust_acct_mast			lnk_customer_account_product
create_date	7	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast
address_type	8	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast_address
line1	9	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast_address
line2	10	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast_address
line3	11	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast_address

conformed business key

Step 3: Apply remapping where necessary

- src_to_target

source_column	column_order	source_table	dv_business_key	dv_bkeygroup	dv_target_table
account_number	1	sapbw_cust_acct_mast	account_id		hub_account
account_pk	2	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast
account_number_second	3	sapbw_cust_acct_mast	account_id	bkgrp_sapbw_second	hub_account
customer_id	4	sapbw_cust_acct_mast	party_id		hub_party
cust_pk	5	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast
prod_code	6	sapbw_cust_acct_mast			lnk_customer_account_product
create_date	7	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast
address_type	8	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast_address
line1	9	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast_address
line2	10	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast_address
line3	11	sapbw_cust_acct_mast			sat_sapbw_cust_acct_mast_address

ties to dv_to_dv.remap_source

*Note the value under dv_bkeygroup, **"bkgrp_sapbw_second"**.*

- dv_to_dv

parenttable	childtable	remapped_hashkey	remap_source
hub_party	lnk_customer_account_product		
hub_account	lnk_customer_account_product		
hub_account	lnk_customer_account_product	dv_hashkey_hub_account_second	bkgrp_sapbw_second
lnk_customer_account_product	sat_sapbw_cust_acct_mast		
hub_party	sat_sapbw_cust_acct_mast_address		

same hub table

The common value *ties* the logic together between tabs, *see **"bkgrp_sapbw_second"** above*. This means that in the link table we will expect to see two surrogate hash key columns for hub_account, the templated "dv_hashkey_hub_account" (based on 'dv_hashkey_${hub_tablename}') and "dv_hashkey_hub_account_second" (remapped)

Why did we need this?
The link table depicted will inherit the parent hub-hash-key names, that's how we promote easy automation, but since there cannot be two dv_hashkey_hub_account columns in a table, one of the hash key columns must be renamed. And why did we not just directly apply that remapping under the src_to_target tab? We could have been dealing with a composite key and therefore it made more sense to assign an arbitrary value under src_to_target tab and control the hub key order under the dv_to_dv tab.

Step 4: Are there dependent child keys?

- src_to_target

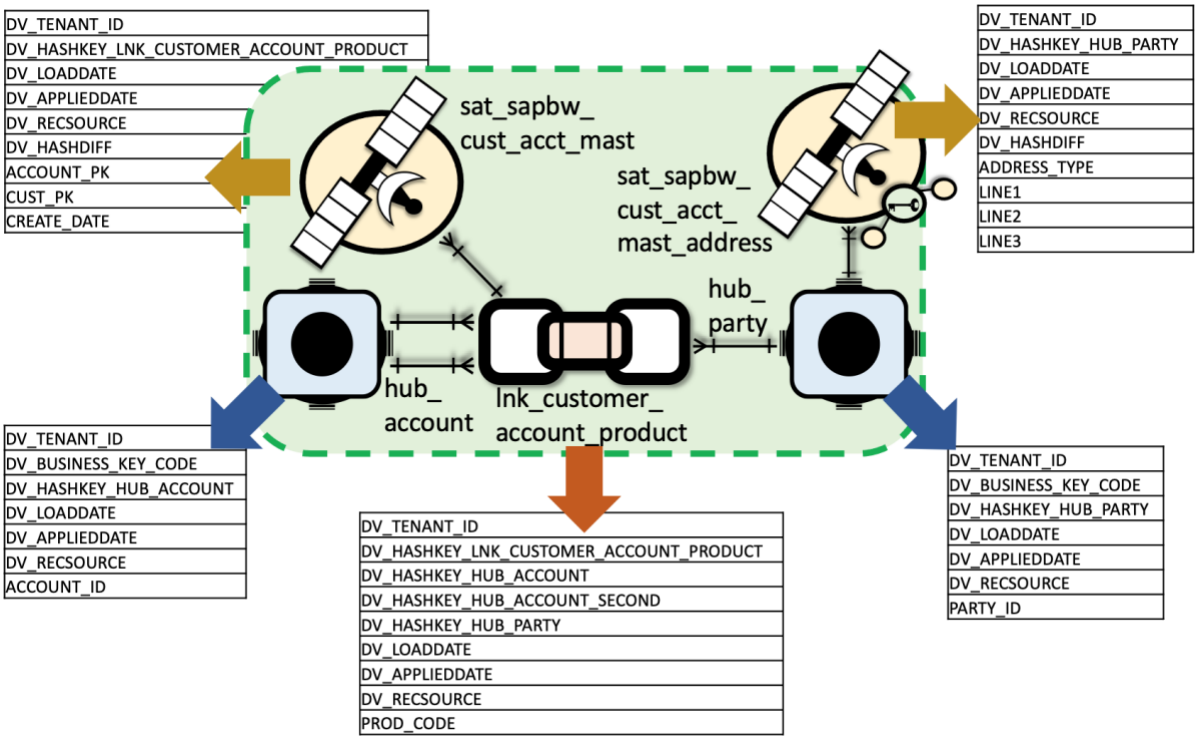
source_column	column_order	source_table	dv_dependent child_key	dv_target_table
account_number	1	sapbw_cust_acct_mast		hub_account
account_pk	2	sapbw_cust_acct_mast		sat_sapbw_cust_acct_mast
account_number_second	3	sapbw_cust_acct_mast		hub_account
customer_id	4	sapbw_cust_acct_mast		hub_party
cust_pk	5	sapbw_cust_acct_mast		sat_sapbw_cust_acct_mast
prod_code	6	sapbw_cust_acct_mast	Y	lnk_customer_account_product
create_date	7	sapbw_cust_acct_mast		sat_sapbw_cust_acct_mast
address_type	8	sapbw_cust_acct_mast	Y	sat_sapbw_cust_acct_mast_address
line1	9	sapbw_cust_acct_mast		sat_sapbw_cust_acct_mast_address
line2	10	sapbw_cust_acct_mast		sat_sapbw_cust_acct_mast_address
line3	11	sapbw_cust_acct_mast		sat_sapbw_cust_acct_mast_address

prod_code and address_type

The only reason the link table appears under **src_to_target.dv_target_table** is to map degenerate dimensions to it, otherwise links should only ever appear under dv_to_dv tab.

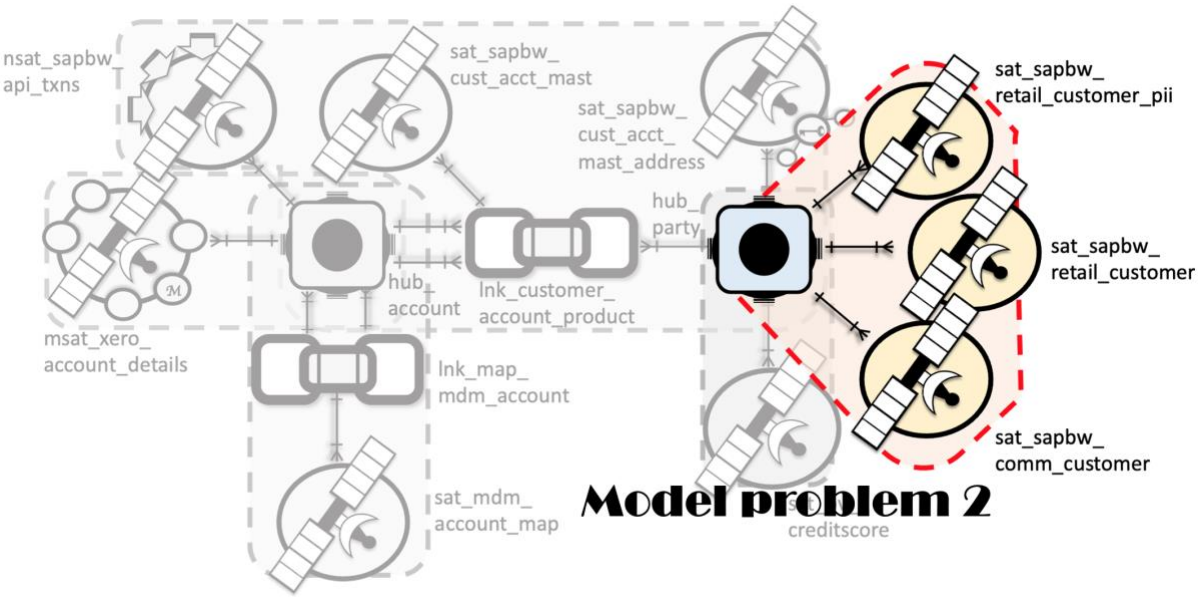
- prod_code goes to the link lnk_customer_account_product
- address_type goes to the satellite sat_sapbw_cust_acct_mast_address

Model data definition outcome:



Satellite tables typically have way more attributes from source, not the four surrogate hash key columns in the link table

Modelling problem 2: Src: sapbw_retail_customer and sapbw_comm_customer



This portion of the model demonstrates ***satellite splitting***, the result of which condenses each satellite by uniqueness of the load. The other call out is the integrated use of the same hub table with business keys coming from the same source whose business keys will never clash. We will also have a single dependent-child key ***go to two satellite tables***.

Step 1: Map source to Target tables and mapping the related artefacts

- src_to_target

source_column	column_order	source_table	dv_target_table	dv_table type
customer_id	1	sapbw_retail_customer	hub_party	
active_details	2	sapbw_retail_customer	sat_sapbw_retail_customer	
first_name	3	sapbw_retail_customer	sat_sapbw_retail_customer_pii	sat
last_name	4	sapbw_retail_customer	sat_sapbw_retail_customer_pii	sat
dob	5	sapbw_retail_customer	sat_sapbw_retail_customer_pii	sat
active	6	sapbw_retail_customer	sat_sapbw_retail_customer	sat
email	7	sapbw_retail_customer	sat_sapbw_retail_customer	sat
customer_id	1	sapbw_comm_customer	hub_party	
name	2	sapbw_comm_customer	sat_sapbw_comm_customer	sat
type	3	sapbw_comm_customer	sat_sapbw_comm_customer	sat
id	4	sapbw_comm_customer	sat_sapbw_comm_customer	sat

- dv_to_dv

parenttable	childtable	source_table
hub_party	sat_sapbw_retail_customer_pii	sapbw_retail_customer
hub_party	sat_sapbw_retail_customer	sapbw_retail_customer
hub_party	sat_sapbw_comm_customer	sapbw_comm_customer

Step 2: Identify and map the business keys

- src_to_target

source_column	column_order	source_table	dv_business_key	dv_business_key_order	dv_target_table
customer_id	1	sapbw_retail_customer	party_id	1	hub_party
active_details	2	sapbw_retail_customer			sat_sapbw_retail_customer
first_name	3	sapbw_retail_customer			sat_sapbw_retail_customer_pii
last_name	4	sapbw_retail_customer			sat_sapbw_retail_customer_pii
dob	5	sapbw_retail_customer			sat_sapbw_retail_customer_pii
active	6	sapbw_retail_customer			sat_sapbw_retail_customer
email	7	sapbw_retail_customer			sat_sapbw_retail_customer
customer_id	1	sapbw_comm_customer	party_id	1	hub_party
name	2	sapbw_comm_customer			sat_sapbw_comm_customer
type	3	sapbw_comm_customer			sat_sapbw_comm_customer
id	4	sapbw_comm_customer			sat_sapbw_comm_customer

Step 3: Are there dependent child keys?

- src_to_target

source_column	source_table	dv_dependent child_key	dv_bkeygroup	dv_target_table
customer_id	sapbw_retail_customer			hub_party
active_details	sapbw_retail_customer	Y	share_depkey_sapbw1	sat_sapbw_retail_customer
first_name	sapbw_retail_customer			sat_sapbw_retail_customer_pii
last_name	sapbw_retail_customer			sat_sapbw_retail_customer_pii
dob	sapbw_retail_customer			sat_sapbw_retail_customer_pii
active	sapbw_retail_customer			sat_sapbw_retail_customer
email	sapbw_retail_customer			sat_sapbw_retail_customer
customer_id	sapbw_comm_customer			hub_party
name	sapbw_comm_customer			sat_sapbw_comm_customer
type	sapbw_comm_customer			sat_sapbw_comm_customer
id	sapbw_comm_customer			sat_sapbw_comm_customer

ties to dv_to_dv.dep_key_shared

Note the arbitrary value in this tab identifying the dependent-child key, ***"share_depkey_sapbw1"***.

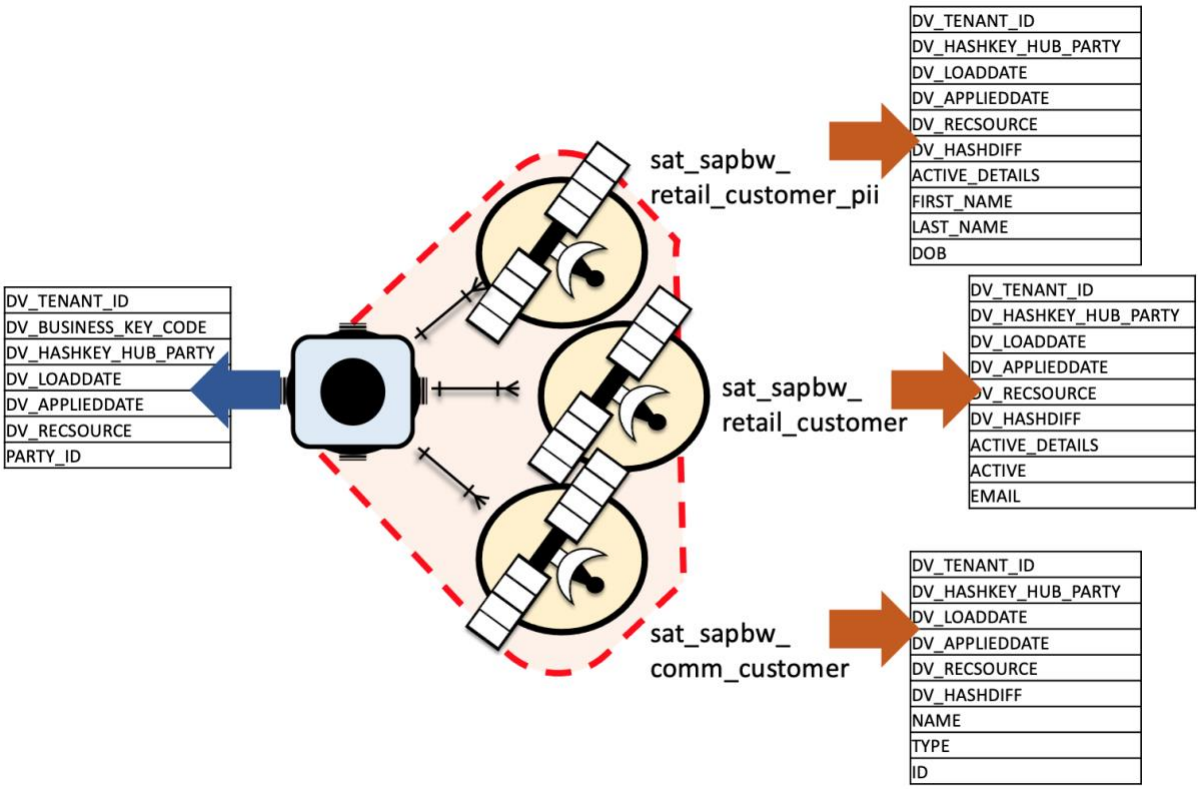
- dv_to_dv

parenttable	childtable	source_table	dep_key_shared
hub_party	sat_sapbw_retail_customer_pii	sapbw_retail_customer	share_depkey_sapbw1
hub_party	sat_sapbw_retail_customer	sapbw_retail_customer	share_depkey_sapbw1
hub_party	sat_sapbw_comm_customer	sapbw_comm_customer	

The common value *ties* the logic together between tabs, ***"share_depkey_sapbw1"***.

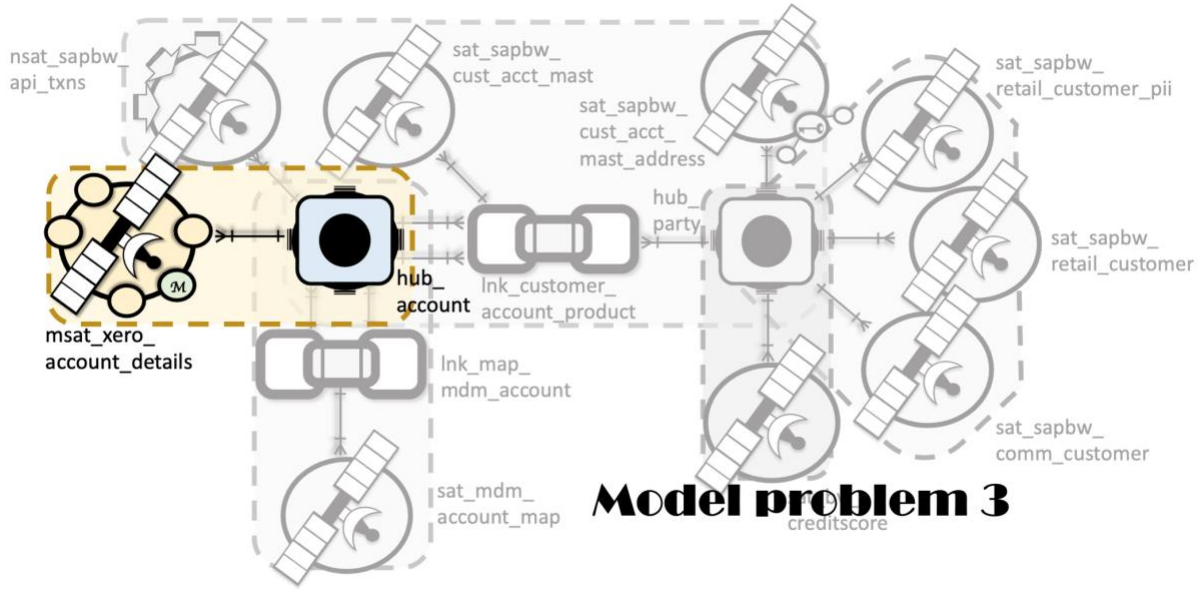
Mapping is required in this instance because we need a way to tie the one or many dependent-child keys to each related satellite that column needs to map to. The value of **src_to_target.dv_target_table** does not matter in this instance.

Model data definition outcome:



Notice how “active_details” appears in both retail customer satellite tables.

Modelling problem 3: Src: xero_account_details



This becomes interesting and an *advanced topic*! Multi-active satellites differ to satellites with dependent-child keys in a few respects. Multi-active SETs are one or more records that make up an ***active SET*** for the parent key, hub or link. A multi-active satellite has an automated column added that arbitrarily counts the records in the set. It is of no value to anyone using this satellite table only and is used to ensure that the table structure stands up uniquely when an index is applied to it. In other words, *never use this new sequence key* to tie it to the single record row! When the record is superseded by a change in the SET an entirely

new sequential value may be assigned to that same record, that may or may not have changed in the SET! Like surrogate hash-keys, do not expose this column to the business!
Through our (pretend) data profiling we have discovered the following:

1. Xero account id will clash with the existing account ids in the hub_account table. A given value will represent a different business object. *We need a prefix, a business key tie-breaker value*.
2. The business key from Xero is *case sensitive*. (Pretend scenario) we need to apply a *different* business key treatment to the standard key treatment or risk collapsing multiple business objects into one and therefore relating un-associated records to the wrong business object.

Let's see how to deal with this!

Step 1: Map source to Target tables and mapping the related artefacts

- src_to_target

source_column	column_order	source_table	dv_target_table	dv_table type
___acct_no___	1	xero_account_details	hub_account	
balance	2	xero_account_details	msat_xero_account_details	msat
fees	3	xero_account_details	msat_xero_account_details	msat
disb	4	xero_account_details	msat_xero_account_details	msat
interest	5	xero_account_details	msat_xero_account_details	msat

- dv_to_dv

parenttable	childtable	source_table
hub_account	msat_xero_account_details	xero_account_details

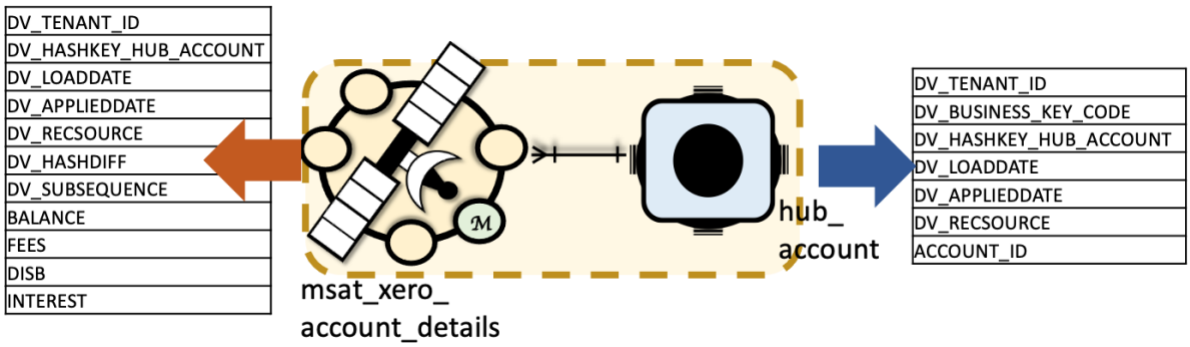
Step 2: Identify and map the business keys

- src_to_target

source_column	source_table	dv_business_key	dv_target_table	dv_business_key_code	dv_business_key_treatments
___acct_no___	xero_account_details	account_id	hub_account	XERO	5
balance	xero_account_details		msat_xero_account_details		
fees	xero_account_details		msat_xero_account_details		
disb	xero_account_details		msat_xero_account_details		
interest	xero_account_details		msat_xero_account_details		

- business key collision code is 'XERO', (see: bit.ly/371PykS)
- business key treatment is 5 (see: bit.ly/3BIziSh)

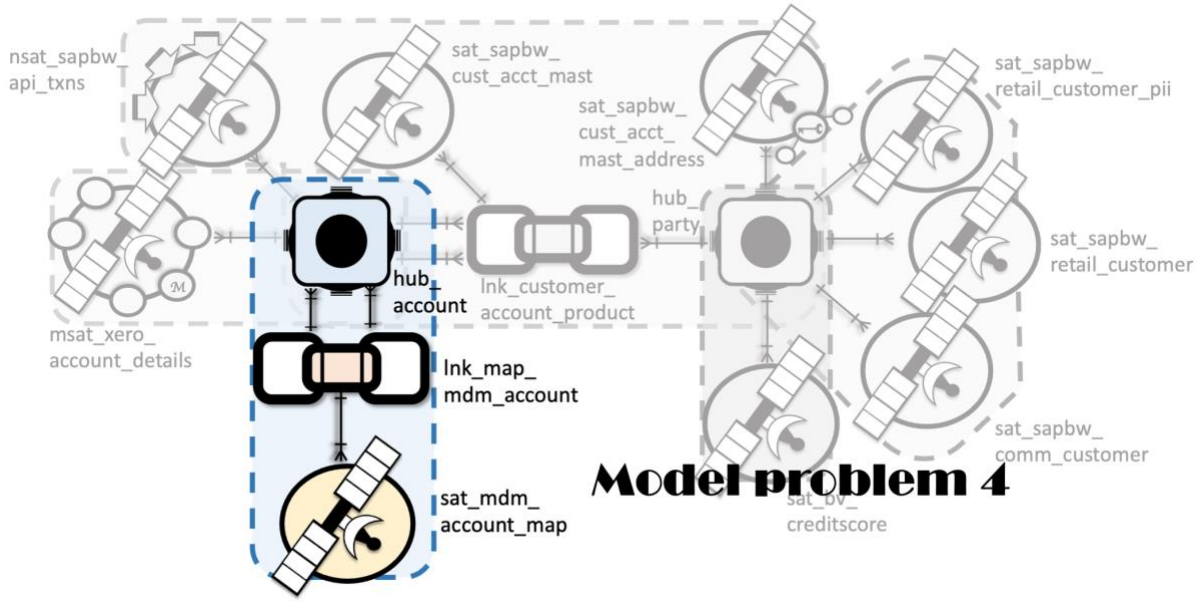
Model data definition outcome:



The Multi-Active Satellite includes an additional column, an arbitrary count under the column DV_SUBSEQUENCE. Also, in this example we needed to:

- Make us of the business key collision code, and
- Use a non-default business key treatment

Modelling problem 4: Src: mdm_account_map



An ***overloaded business key column*** could be provided as is the case of source system business key to MDM id mapping. Now you could attempt to split this out, but every new split requires coding effort to define what that is. A more optimal and robust approach is to include that ability to dynamically assign the business key collision code where needed within the staged column itself. MDM will provide an account source column, again do not default to treat this column as the business key collision code itself, instead develop a map between source-column value and the business key collision code, include an ***error trap*** when a new source is detected and simply update the mapping to resolve it. What will happen now is that a column will be added in staging denoting what that business key collision code should be per business key from source.
This solution is a hybrid of

- ***data driven modelling*** – the mapping table ensures the same code to populate the staged column is driven by a lookup table.
- ***model driven modelling*** – the staged table itself contains a column that defines a portion of the model outcome.

Seems complicated but applying this technique and controlling it though the Excel approach is quite simple.

Step 1: Map source to Target tables and mapping the related artefacts

- src_to_target

source_column	column_order	source_table	dv_target_table	dv_table type
src_id	1	mdm_account_map	hub_account	
mdm_id	2	mdm_account_map	hub_account	
dv_business_key_code	3	mdm_account_map	sat_mdm_account_map	sat
create_date	4	mdm_account_map	sat_mdm_account_map	sat
match_rule	5	mdm_account_map	sat_mdm_account_map	sat
match_type	6	mdm_account_map	sat_mdm_account_map	sat
confidence_level	7	mdm_account_map	sat_mdm_account_map	sat

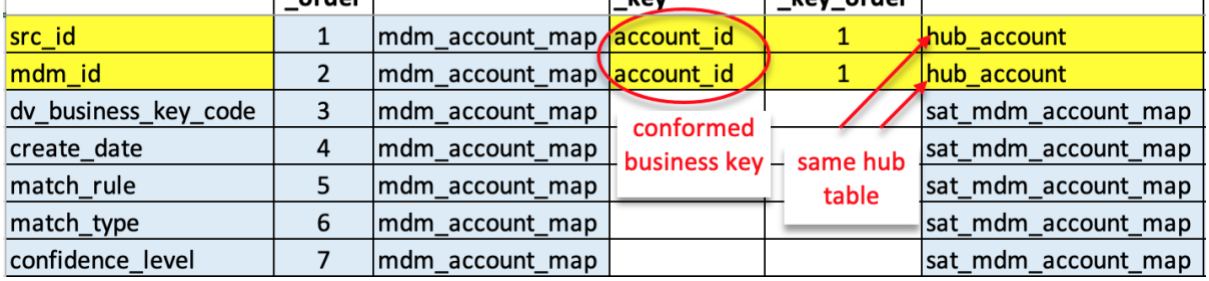
- dv_to_dv

parenttable	childtable	source_table
hub_account	lnk_map_mdm_account	mdm_account_map
hub_account	lnk_map_mdm_account	mdm_account_map
lnk_map_mdm_account	sat_mdm_account_map	mdm_account_map

Step 2: Identify and map the business keys

- src_to_target

source_column	column_order	source_table	dv_business_key	dv_business_key_order	dv_target_table
src_id	1	mdm_account_map	account_id	1	hub_account
mdm_id	2	mdm_account_map	account_id	1	hub_account
dv_business_key_code	3	mdm_account_map			sat_mdm_account_map
create_date	4	mdm_account_map			sat_mdm_account_map
match_rule	5	mdm_account_map			sat_mdm_account_map
match_type	6	mdm_account_map			sat_mdm_account_map
confidence_level	7	mdm_account_map			sat_mdm_account_map



Step 3: Apply remapping where necessary

- src_to_target

source_column	source_table	dv_business_key	dv_business_key_order	dv_bkeygroup	dv_target_table
src_id	mdm_account_map	account_id	1		hub_account
mdm_id	mdm_account_map	account_id	1	bkgrp_mdm_id	hub_account
dv_business_key_code	mdm_account_map				sat_mdm_account_map
create_date	mdm_account_map				sat_mdm_account_map
match_rule	mdm_account_map				sat_mdm_account_map
match_type	mdm_account_map				sat_mdm_account_map
confidence_level	mdm_account_map				sat_mdm_account_map

arbitrary value to to dv_to_dv. remap_source

"bkgrp_mdm_id" to tie column to remap

- dv_to_dv

parenttable	childtable	remapped_hashkey	remap_source
hub_account	lnk_map_mdm_account		
hub_account	lnk_map_mdm_account	dv_hashkey_hub_mdm_account	bkgrp_mdm_id
lnk_map_mdm_account	sat_mdm_account_map		

same hub

"bkgrp_mdm_id" to tie column to remap/override what would have been mapped base on "dv_hashkey_${hub_tablename}" template

Step 4: Are there row assignments needed?

- src_to_target

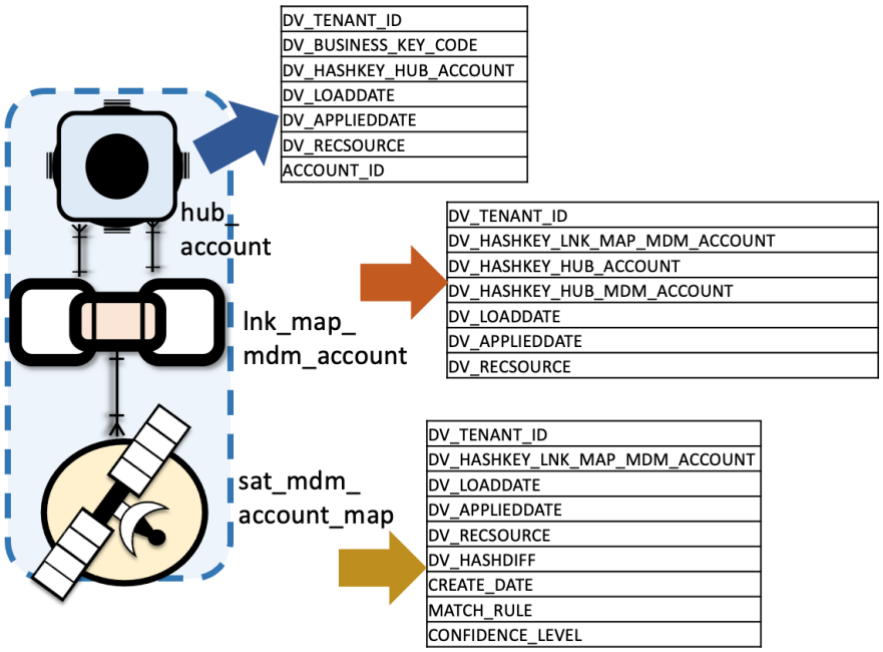
source_column	dv_business_key	dv_target_table	dv_tag	dv_tag_group	dv
src_id	account_id	hub_account		MDM_ACCOUNT_MAP_GRP	
mdm_id	account_id	hub_account			
dv_business_key_code		sat_mdm_account_map	DV_BKEYCOLCODE	MDM_ACCOUNT_MAP_GRP	
create_date		sat_mdm_account_map			
match_rule		sat_mdm_account_map			
match_type		sat_mdm_account_map			
confidence_level		sat_mdm_account_map			

identifies a BKCC

use the **value** in this column to apply the collision code to this **business key** column

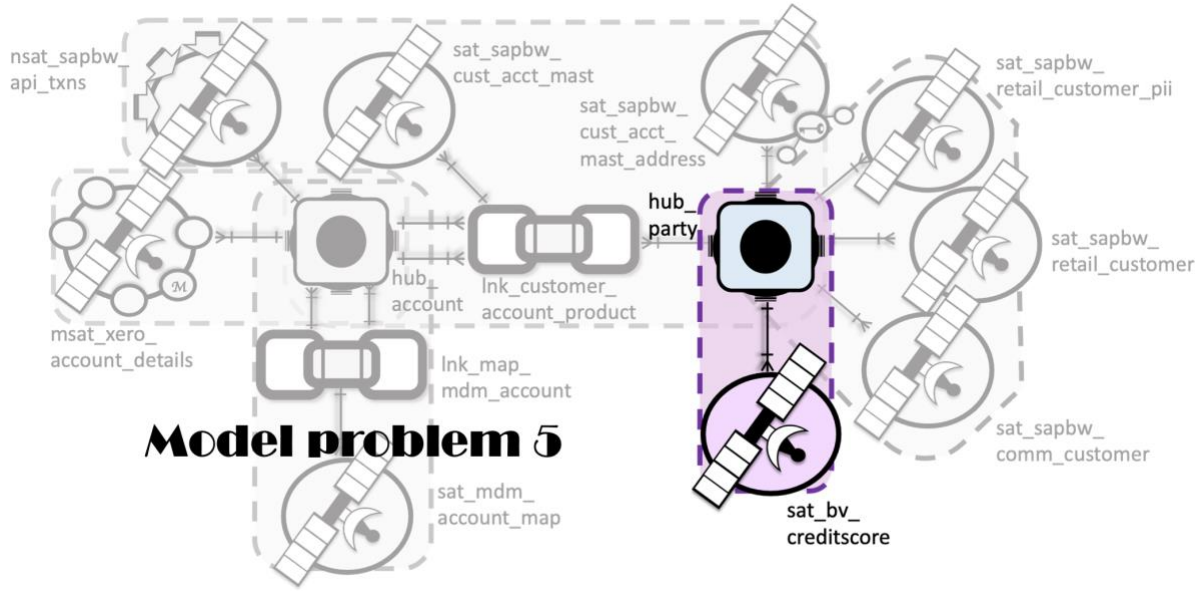
In the staged file itself we have a column that contains the outcome of the mapping called **dv_business_key_code**. The name of this column does not matter because under the dv_tag configuration we have identified that column to contain the row-level business key collision code. Now the next column **dv_tag_group** is necessary because we need to know which business key (or keys) this collision code applies to. An arbitrary unique value ties these two together, src_id (contains the source business key) must have a row-level business key collision code value applied to it.

Model data definition outcome:



The row-level collision code mapping is necessary because of an overloaded business key column,

Modelling problem 5: Src: bv_creditscore



Business Vault links or satellites integrate to existing raw vault hub tables, the mapping process is the same but if the data vault built is ***not bitemporal*** then you might have a more difficult time combining the data together for reporting. This is where ***applied date*** comes in, it is the snapshot date of all the applicable data and information about a hub or a link. Future business dates, past business dates, at a point in time have an applicability date. This is the snapshot of the facts as we knew it today!
For derived content that is stored in business vault the applied date ***must match*** the data that was used to derive that content. *The audit trail*. To do this we will reuse the dv_tag concept introduced earlier. Keep in mind that although this is a simplistic example, there really isn't anything stopping you from creating business vault satellites with dependent child keys or

even multi-active satellites! Business vault although ***sparsely modelled*** does use the ***same loading structures*** as raw vault. This might be why practitioners resort to a *virtual* business vault because of not being able to resolve the applicability of derived content. Well, this is the solution.

Step 1: Map source to Target tables and mapping the related artefacts

- src_to_target

source_column	column_order	source_table	dv_target_table	dv_table type
party_id	1	bv_creditscore	hub_party	
dv_applieddate	2	bv_creditscore	sat_bv_creditscore	sat
creditscore	3	bv_creditscore	sat_bv_creditscore	sat
rating_confidence	4	bv_creditscore	sat_bv_creditscore	sat
score_date	5	bv_creditscore	sat_bv_creditscore	sat

- dv_to_dv

parenttable	childtable	source_table
hub_party	sat_bv_creditscore	bv_creditscore

Step 2: Identify and map the business keys

- src_to_target

source_column	column_order	dv_business_key	dv_business_key_order	dv_target_table
party_id	1	party_id	1	hub_party
dv_applieddate	2			sat_bv_creditscore
creditscore	3			sat_bv_creditscore
rating_confidence	4			sat_bv_creditscore
score_date	5			sat_bv_creditscore

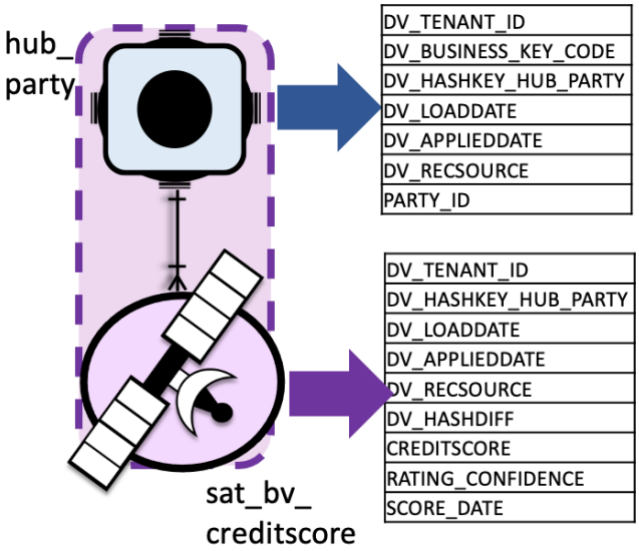
Step 3: Are there row assignments needed?

- src_to_target

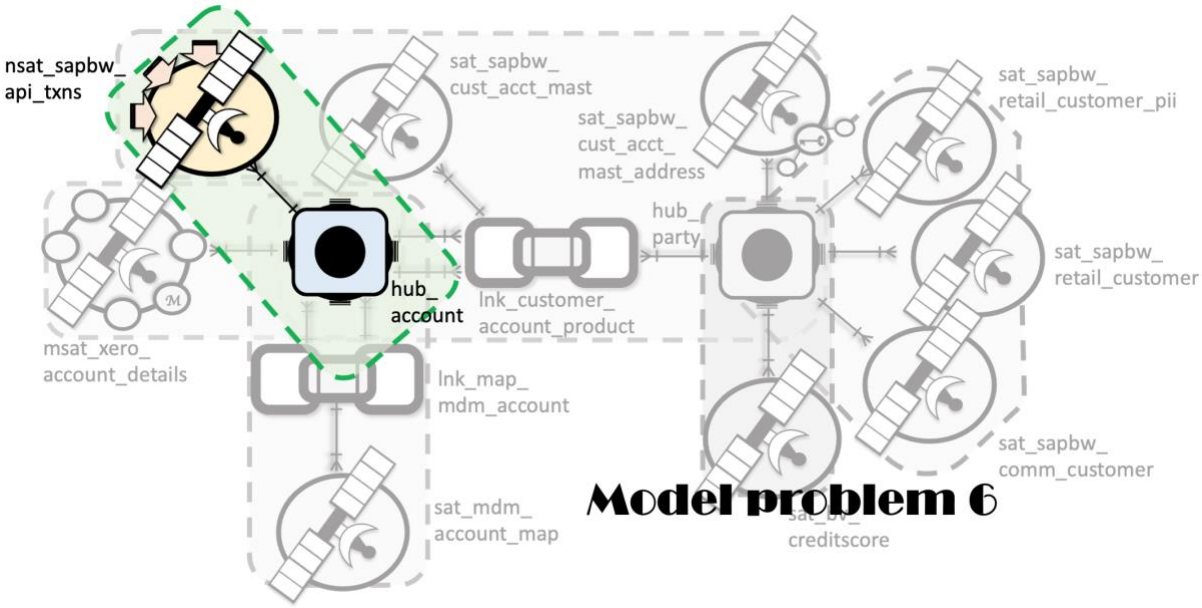
source_column	source_table	dv_target_table	dv_tag	dv
party_id	bv_creditscore	hub_party		
dv_applieddate	bv_creditscore	sat_bv_creditscore	DV_APPLIEDDATE	
creditscore	bv_creditscore	sat_bv_creditscore		
rating_confidence	bv_creditscore	sat_bv_creditscore	identifies the column carrying the applied date	
score_date	bv_creditscore	sat_bv_creditscore		

Like the example before this one, the column name does not matter, it is the assignment as a dv_tag that identifies which column to use as the applied date. ***It will never be the load date*** because the load date is a value denoting *when* the record was loaded into data vault. *Never* manipulate this date! The load date itself, can act a version date if multiple versions of the same records are loaded with the same applied date. Even better this concept can be used in [Data Vault's XTS pattern.]

Model data definition outcome:



Modelling problem 6: Src: sapbw_api_txns



Finally, how do we map non-historised content?

Step 1: Map source to Target tables and mapping the related artefacts

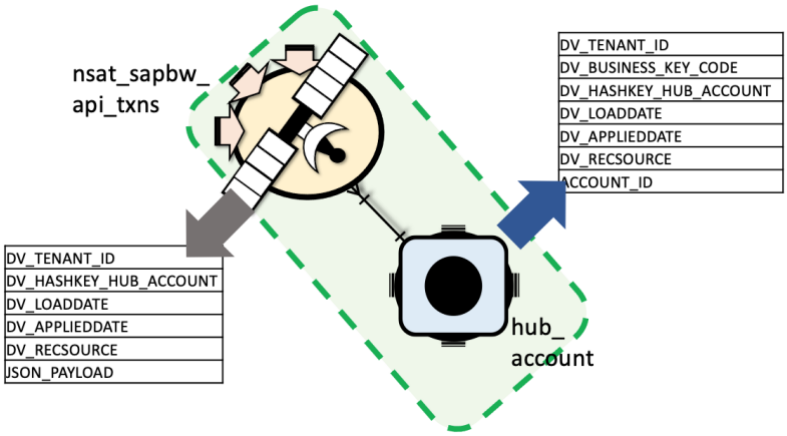
- src_to_target

source_column	column_order	source_table	dv_business_key	dv_business_key_order	dv_target_table	dv_table type
acc	1	sapbw_api_txns	account_id	1	hub_account	
json_payload	2	sapbw_api_txns			nsat_sapbw_api_txn	nsat

- dv_to_dv

parenttable	childtable	source_table
hub_account	nsat_sapbw_api_txns	sapbw_api_txns

Model data definition outcome:



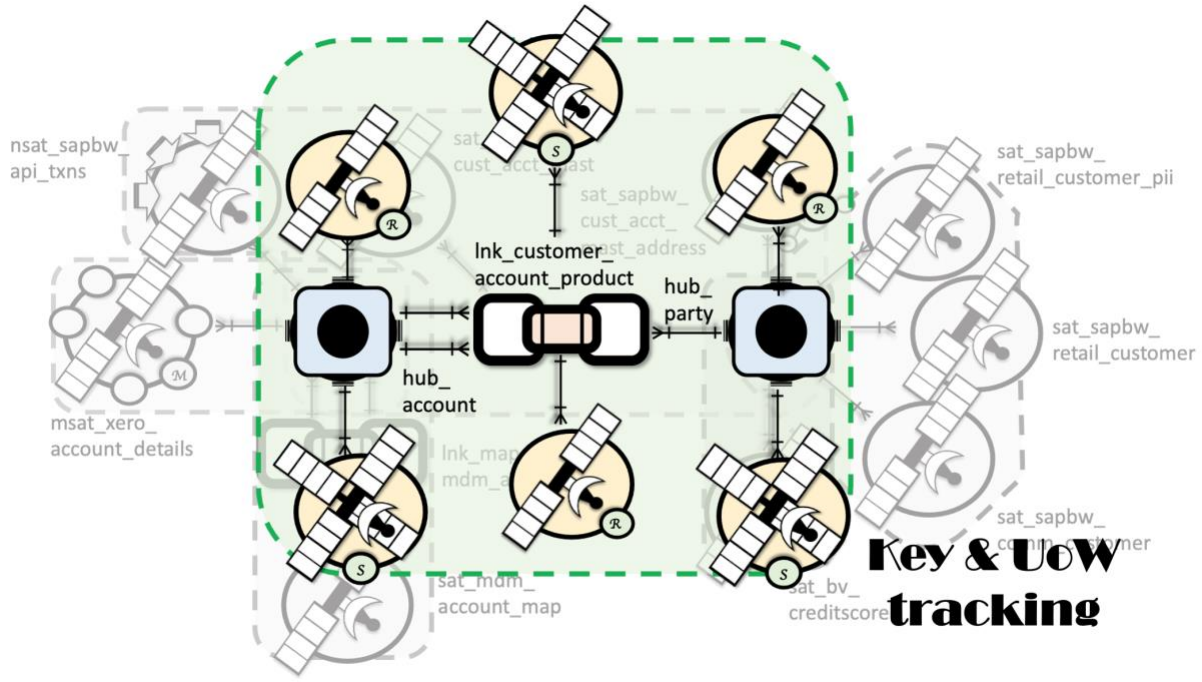
Notice the lack of a record hash….

Tracking Business Keys and Relationships

For those keeping track, yes there are additional ***auxiliary*** satellites we can enable in data vault. They are not based on the descriptive content per se, but rather the status and appearance of business objects and their relationships.

First up,

Record and Status Tracking



Step 1: Map source to Target tables and mapping the related artefacts

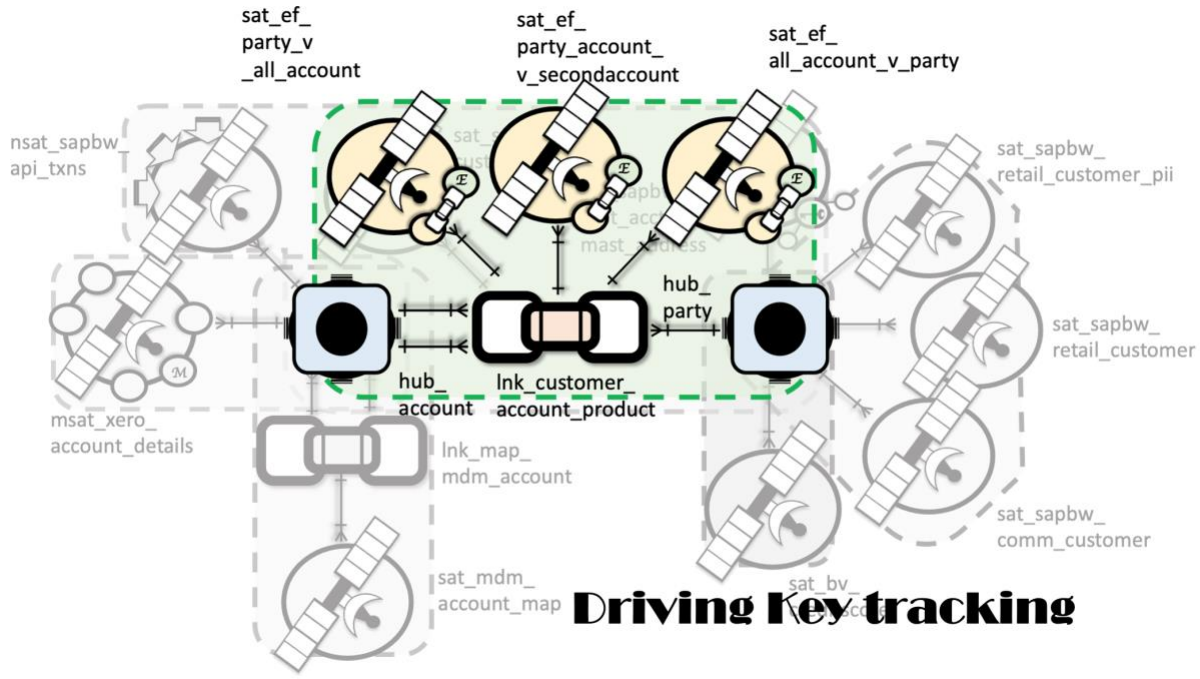
- dv_to_dv

parenttable	childtable	dv_assign_recordtracking	dv_assign_statustracking
hub_party	lnk_customer_account_product	satr_hub_party	sats_hub_party
hub_account	lnk_customer_account_product	satr_hub_account	sats_hub_account
hub_account	lnk_customer_account_product		
lnk_customer_account_product	sat_sapbw_cust_acct_mast	satr_lnk_customer_account_product	sats_lnk_customer_account_product
hub_party	sat_sapbw_cust_acct_mast_address		

An extreme example but you could deploy a record and status tracking satellite for each business key and relationship in your source, but consider the following:

- what is the chief reason for tracking? Don't deploy these satellites if there is no reason to, after all *nothing in life is free*.
- what is the column that is expected to have the most complete list of accounts or parties? For example, would there be a reason to track a column that optionally contains a business key in a relationship? Maybe, you decide.
- If we choose to use data vault table names in satellite table name construction that means additional sources integrating to the same hub or link would either need its own record and/or status tracking satellite (maybe concatenate source and target table names) or the same satellite can load all source-based business key and relationship tracking. The difference is in the **record source column**. This integration works because the record or status tracking table structure is always the same.
- Derived status tracking only works if the data loaded from source is a full snapshot each day, and we detect inserts, updates and deletes.

Effectivity satellite



In the absence of a **business date** tracking relationship changes, or if you need to track effectivity differently to how the source tracks it, we have the **driver key** and **effectivity satellite** concept. To support the desired scalability an additional tab is needed.

- Driver key is party_id

hub_table	lnk_table	source_table	eff_sat
hub_party	lnk_customer_account_product	sapbw_cust_acct_mast	sat_ef_party_v_all_account

- Driver key is party_id and account_id

hub_table	lnk_table	source_table	eff_sat
hub_party	lnk_customer_account_product	sapbw_cust_acct_mast	sat_ef_party_account_v_secondaccount
hub_account	lnk_customer_account_product	sapbw_cust_acct_mast	sat_ef_party_account_v_secondaccount

- Driver key is account_id and the second account_id

hub_table	lnk_table	source_table	remapped_hashkey	eff_sat
hub_account	lnk_customer_account_product	sapbw_cust_acct_mast		sat_ef_all_account_v_party
hub_account	lnk_customer_account_product	sapbw_cust_acct_mast	bkgrp_sapbw_second	sat_ef_all_account_v_party

To peruse this mapping take the attached Excel for a spin (link below).

Final words

No code is supplied in these examples, customers have used a similar approach backed by python to generate the required YAML, I have designed and advised on this approach for a few customers already, one was even using SAS to generate the necessary SQL! This Excel approach illustrated a mapping structure that has worked, ultimately with a culture of open

programming and collaboration led to any identified shortcomings being easily resolved. Of course, any automation tool even developed in-house should look to at some point to replace this frontend with a GUI that includes the appropriate constraints to automate the model at scale and be appropriately governed! And even backed look to a more scalable solution managed by a Schema Vault.
For customers wanting to use a commercial automation tool, this Excel mapping approach is not needed.
Also look to include automated testing and dashboarding ensuring the models deployed are fault free and maintain confidence in the solution.
One thing is clear from design and implementation perspective… when embarking on a data vault implementation there is an element of data modelling **and** data engineering, (from my experience) not all data engineers make great data modellers; data modellers must be left in charge of what the model will look like, data engineers must engineer and automate the data vault patterns. It's why you might see engineers deploy source-based business key collision codes or attempt to add effectivity columns to link tables! Neither of which will scale and inevitably lead to a new legacy data warehouse!

To get a hold of more of this type of deep dive on pragmatic approaches into building a data vault, refer to the Data Vault Guru!

Excel map itself is available here: ?? Github ??

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.