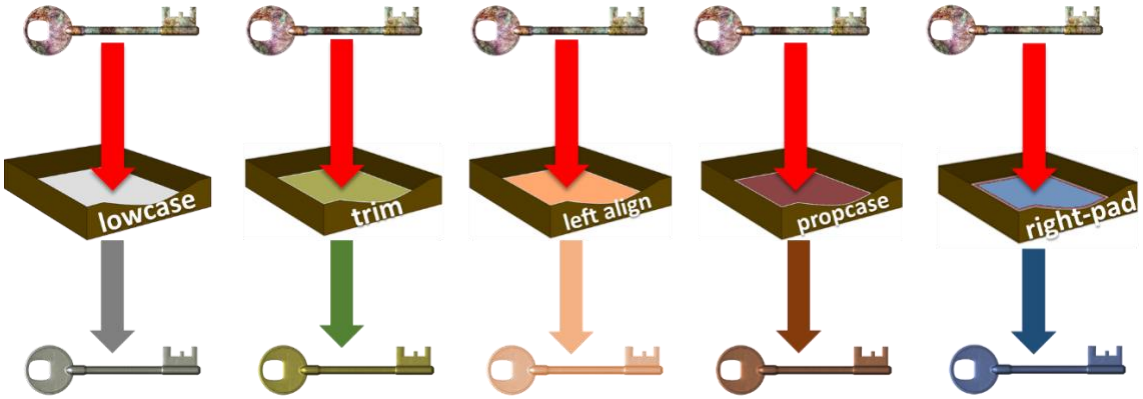
Business keys are the crux of a data vault model; these are the unique values used to identify a business entity. The business entity could be an account, a product, a customer, that order number you received after making a purchase, that insurance policy number you were assigned when you have successfully applied for accident insurance, the VIN number found on a motor vehicle. Details about those business entities are tracked; such as details in insurance premiums, residential address and changes thereof. If you do not have a business key, you do not have a data vault.

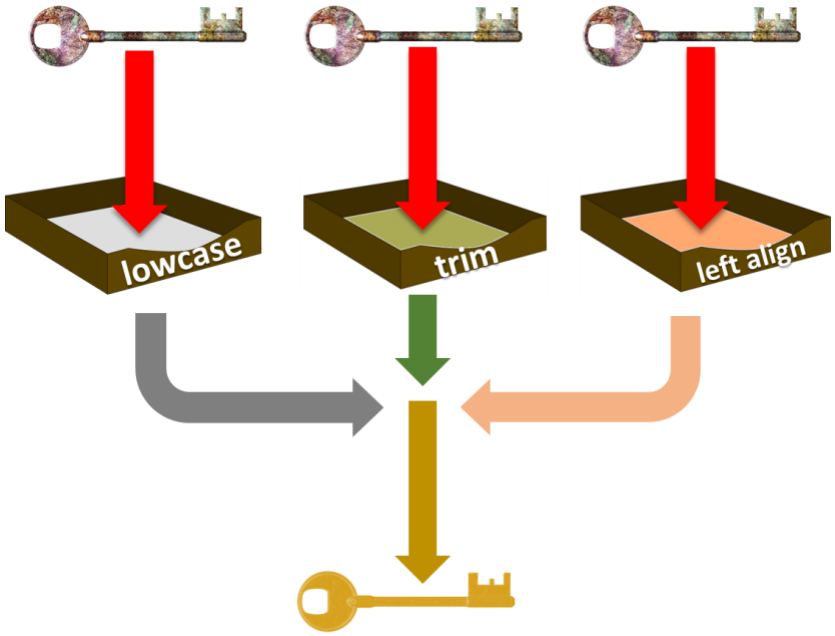
So that is business keys squared away.
What are treatments then? In the statistical world these are the actions we apply to a value to give it more context or change its value; in other words, applying an action to influence a variable with justification. In credit risk the word is used to define the actions applied to manage risk. In database marketing we apply the terminology to describe the offers we make to a prospect.
When applying treatments to business keys, each function we apply to manipulate the business key can be thought of as an action with justification.



The Default Treatment
In data vault 2.0 business keys are given a default treatment as input to a hash function to produce a surrogate key. These surrogates and related business key(s) will behave consistently to honour the basic premise of surrogate key usage. If we have an insurance policy hub, expect to see a hash-key column that contains the hashed version of the business key(s) after applying the default treatments.

The default treatments applied in data vault 2.0 are:

Treatment	Description and example
Lowcase/Upcase	Applies lower casing to the text string
Trim	Removes trailing blanks after the text string
Left-align	Removes the leading blanks before the text string



Why do we treat business keys in this way?

It relates to how the source systems have defined the business key types. When systems use numeric keys as a sequence, treatments are straight forward. However business keys with and alphanumeric data type pose more challenges. What if two systems have business keys that represent the same entity, but integrate their keys by applying an upper or lower function on ingestion? If one system applies upper and the other lower, then they would not integrate into the same hub. Applying the default treatments however ensures that sticky finger issues (manual entries) are catered for and provides a suitable clean-up for key consistency and integration. The other thing to consider is that downstream software such as Excel are not case sensitive, therefore having two or more entries in the data vault for a single business entity introduces technical debt into the data vault.

What if the source system has a problem?

Although this should be a rare occurrence, it is unfortunately a reality. A source system may contain business keys implemented with case sensitivity. If the case differs in the business key, they are two different business keys related to different descriptive details. The reasons for implementations like this may be due to

- poorly defined business requirements for the source system,
- an attempt to divide two business entities who happen to have registered similar business keys,
- test data loaded into a production environment,
- manually entered business keys with no domain constraints in the source system,
- old system data migration efforts,
- and more!

Before we try to solve the problem in data vault, the question needs to be asked of the source system to fix this problem. This issue is not only a business intelligence problem but a significant integration and data quality issue that will continue to erode business efficiency.

Unfortunately, there are source systems that have this business key debt. We must therefore have a strategy to deal with this issue in data vault if the issue cannot be resolved. It is not the only sticking point though. Reference data may present similar challenges if the same hub loading patterns are used to load reference hubs.

Modelling reference tables into data vault 2.0?

If you have decided to model reference tables into data vault as hubs and satellites, then there may have an inherent problem with applying the default business key treatments to the business key column. If 'lowcase' is applied to the reference data business key, then the query or (possible even join code) will need to apply the same function to utilise the lookup code value when resolving at time of data consumption. Those that use the lookup code explicitly will now suddenly need to be cognisant of these nuances. And what if the case, leading blanks or trailing blanks in the lookup code means something different if they are changed!

But why model reference tables as hubs and satellites? They naturally inherit the ingestion patterns already developed when loading hubs and satellites; and changes to lookup content are naturally captured in data vault without introducing a new ingestion pattern. Note, avoid creating a link sprawl by creating a link table for every lookup code in a satellite mapping to a hub-reference table should be avoided as this means you would have snowflaked a satellite and a satellite cannot be related to more than one hub or link.

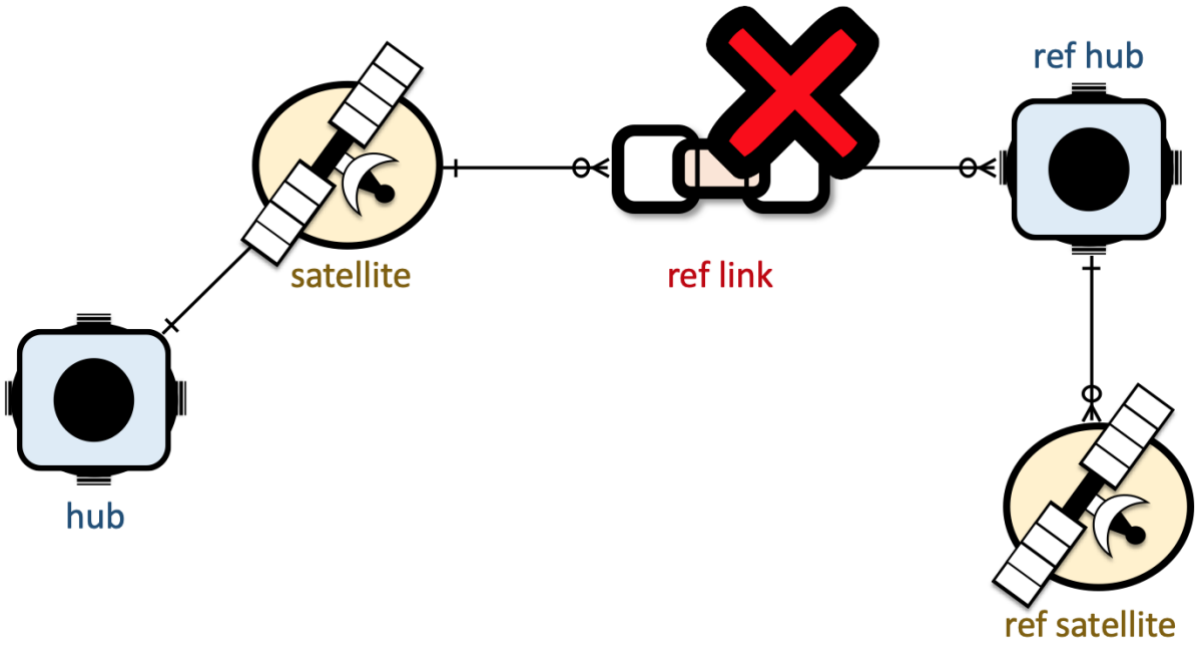


Figure 1 Not a vault!

The solution

What if we can provide a configurable method of applying specific treatments suitable for the situation without having to change any code? We will use a bit of math (pun intended but more obvious why that is later) to create a dynamic way of assigning non-default treatments on a business key.

First let's lay the math

#	Treatment	Exponent $2^{(n-1)}$	Calculated value	Excel formula

1	Lowcase	$2^{(1-1)}$	1	2^1 or POWER(2, 1-1)
2	Trim	$2^{(2-1)}$	2	2^2 or POWER(2, 2-1)
3	Left-align	$2^{(3-1)}$	4	2^3 or POWER(2, 3-1)
4	Propcase	$2^{(4-1)}$	8	2^4 or POWER(2, 4-1)
5	Right-pad	$2^{(5-1)}$	16	2^4 or POWER(2, 4-1)

**The informational section of the reference table is in blue.*

Now let's apply it through an example. Let's say we only need the business key to be left-aligned and lowcased, by referring to the table above we pick the calculated values that align to lowcase and left-align and add them together.

f(lowcase) + f(left-align)
= 1 + 4
= 5
So, for this business key we assign the value 5 in the mapping, now let's see how this is applied.

Method 1: looping through the reference table from bottom up

#	Treatment	Calculated value	Steps	Rule	Result	Remainder
1	Lowcase	1	5	Is 1 smaller than / equal to 5?	Yes	0
2	Trim	2	4	Is 2 smaller than / equal to 5?	No	1
3	Left-align	4	3	Is 4 smaller than / equal to 5?	Yes	1
4	Propcase	8	2	Is 8 smaller than / equal to 5?	No	5
5	Right-pad	16	1	Is 16 smaller than / equal to 5?	No	5

**code logic is in green.*

1. The logic works like this, we start from the bottom in step 1 and check if the calculated exponent of 'right-pad' (16) is smaller than or equal to the assigned value of 5. The answer is no so we move up to step 2.
2. Step 2 asks the same question based on the calculated value for 'propcase' (8) and the answer is no so we move up to step 3.
3. Step 3 asks if the calculated value for 'left-align' (4) is smaller than or equal to 5 and the answer is yes. We subtract 4 from 5 and keep the remainder. The positive return to the question means we must apply the 'left-align' treatment and we move up to step 4 (we still have a remainder).
4. Step 4 asks if the calculated value for 'trim' (2) is smaller than or equal to 1 (the remainder). The answer is no and we move up to step 5.
5. Step 5 we ask if the calculated value for 'lowcase' (1) is smaller than or equal to 1 and the answer is yes; therefore, we must also apply the lowcase treatment to the business key and subtract the calculated value from the remainder (1 minus 1) and we end up with zero.

Implemented as code it is a finite loop defined by the number of elements in the reference table

Loop sample	Explanation

bit=5;
 	
do I = 5 to 1 by -1;
  chk=2**(i-1);
 	
  if bit >= chk then do;
  text=put(chk, bitlookup.);
  bit=bit-chk;
  put text=;
  end;
end;
	bit is the assigned value to determine which treatments to apply
	
	Do loop starts from the max number of elements in the lookup. chk is assigned the exponent of the loop number going down from 5 (chk=16) then 4 (chk=8) and so on.
	
	Apply Rule: if bit >= chk, yes then lookup mapping (4 = propcase & 1 = lowcase.
	Update bit (5-4 then 1-1).
	
	DO Loop can be replaced with DO UNTIL bit=0 or DO WHILE bit <> 0.

Method 2: bitwise join

An alternative way to think about this logic is in the way of bits (there you go, there's the pun).

16	8	4	2	1
0	0	1	0	1

= 4 + 1

= 5

Using SQL, the code appears a bit simpler

SQL sample	Explanation
create table sample with val=5 select m.val, o.treatment from sample m cross join bitflag o where m.val & o.bit <> 0	Just a sample table with the selected calculated value to map. Cross join with bitwise '&' will join the bit representation of val with every matching bit in the lookup table. See image below on how this works.

How the bitwise '&' is applied

#	Exponent	16	8	4	2	1
1	1	0	0	0	0	1
2	2	0	0	0	1	0
3	4	0	0	1	0	0
4	8	0	1	0	0	0
5	16	1	0	0	0	0
0	5	0	0	1	0	1

The bitwise '&' will return the matching bits;

Configuration driven modelling

When defining the source to target mapping during modelling, we need to have the business key treatment value applied to that grain. The below being a surrogate of the collected metadata during modelling as example.

Source Table	Source Column	Target Table	Target Column	TREATMENT
CUSTOMER	CUSTOMER_ID	HUB_CUSTOMER	CUSTOMER_ID	5

It is possible to extend the reference table with the decision on whether you want to hash a business key as well. *Why would we be interested in doing this?* If what you are modelling is based on a single column business key, then it could be possible that the ingestion and consumption from the untreated surrogate key could outperform a data vault based on hash keys. It takes time to hash a business key into a hash key (and apply other treatments to a business key); one such use case is the creation of a real-time metric vault under a dashboard with time-travel.

Let's outline reasons for picking either

Pick the business key	**Pick the hash key**
Length is less than a hashed business key value length	Length is less than the business key value length
A single column represents the business key	Composite columns represent the business key
Platform does not support binary data type	Platform supports binary data type

Applying this solution depends if your automation tool can utilize a lookup table to define the business key treatments. Applying business key treatments in this way makes the solution data-driven and thus there is no change to the underlying code when changing treatments. The best part about this method is that if you need to add a new treatment you simply add it to the end of the lookup table and the existing treatment mappings are unaffected. A word of caution; avoid applying functions that change the semantic meaning of the business key as this becomes a business rule and the output should be recorded in a business vault hub.

The above method now provides the flexibility to differentiate treatments for those very small but real use cases where integration efforts within the application platforms within an enterprise have failed.

Final thoughts:

- Always try to resolve the issue at source first! This is the most cost effective and efficient course of action!
- Not applying the default treatments on a business key should be the exception and not the rule.