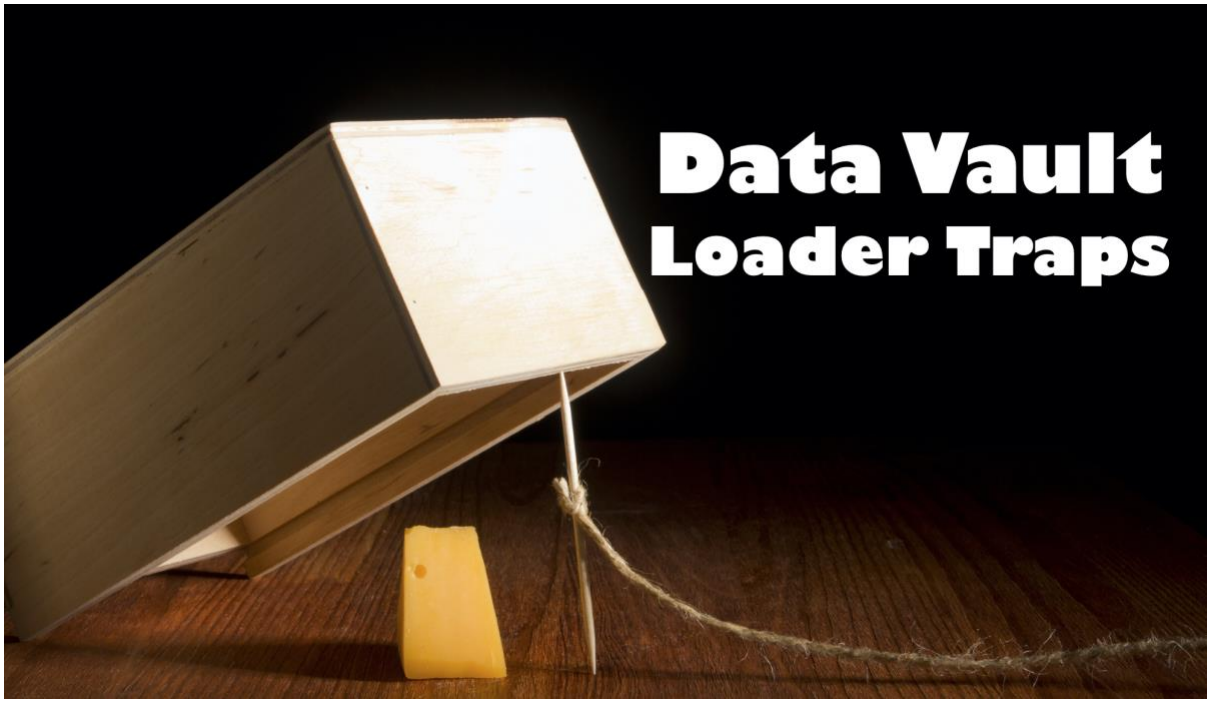


Data Vault Loader Traps

1: Ensure you are comparing against current records for Satellite loads.

In an earlier blog post (see: bit.ly/3dUHPIS) about test automation the scenario was highlighted about getting the load to satellite tables correct. The data vault test suite (available here: bit.ly/3nyYF48) is designed to keep your data vault loading patterns honest. This post will show by way of example where not following this loader rule can go wrong.

Day 1 staged content

Hashed(ACCOUNT_ID)	Satellite attribute	Load-Date
hashed(1)	Original	1-Jan
hashed(2)	Original	1-Jan

Day 1 satellite after staged content has loaded

Hashed(ACCOUNT_ID)	Satellite attribute	Load-Date
hashed(1)	Original	1-Jan
hashed(2)	Original	1-Jan

Day 2 staged content

Hashed(ACCOUNT_ID)	Satellite attribute	Load-Date
hashed(1)	Change	2-Jan
hashed(2)	Original	2-Jan

Day 2 satellite after staged content has loaded, for account id 1 the change is loaded.

{ account_id: 1, attribute: 'Change', loaddate: 2-Jan } differs from
{ account_id: 1, attribute: 'Original', loaddate: 1-Jan }

Hashed(ACCOUNT_ID)	Satellite attribute	Load-Date
hashed(1)	Original	1-Jan
hashed(1)	Change	2-Jan
hashed(2)	Original	1-Jan

Day 3 staged content, the previous state of account id 1 has returned, 'Original.

Hashed(ACCOUNT_ID)	Satellite attribute	Load-Date
hashed(1)	Original	3-Jan
hashed(2)	Original	3-Jan

If we compare for the whole satellite for the account_id you will get the following:

Hashed(ACCOUNT_ID)	Satellite attribute	Load-Date
hashed(1)	Original	1-Jan
hashed(1)	Change	2-Jan
hashed(2)	Original	1-Jan

{ account_id: 1, attribute: 'Original, loaddate: 3-Jan } differs from
{ account_id: 1, attribute: 'Change, loaddate: 2-Jan } but matches
{ account_id: 1, attribute: 'Original', loaddate: 1-Jan }
This is obviously **incorrect** because Account ID:1 should have reverted to 'Original', the outcome should be

Hashed(ACCOUNT_ID)	Satellite attribute	Load-Date
hashed(1)	Original	1-Jan
hashed(1)	Change	2-Jan
hashed(1)	Original	3-Jan
hashed(2)	Original	1-Jan

And no, including the business key in the calculation of HashDiff does not fix this! Instead, the satellite comparison must be between the staged content and the *current* record for an account_id. Now imagine, a link that stores a relationship will also track relationship changes and a relationship can revert to the original relationship! We deal with this pattern using driving keys and effectivity satellites when the source does not supply a business date tracking that very event! See: bit.ly/3vTn1rA

2: Using Multi-Table insert to load to a common hub table will load duplicate records.

Multi-table insert does as advertised (see: bit.ly/3h97Ypz), load target tables in parallel. When modelling a same-as link or a hierarchy link the target tables will be:

- A Link table,

- A Hub table for the business object's first business key and the same hub table for the business object's second business key

The loader for a hub table must ensure that only new business keys are loaded to the target hub table leaving the hub table with the same integrity as before we loaded to it, a unique list of business keys. In a multiple table insert there is a thread for each insert; there is no way to ensure that both threads will consolidate the check to not load duplicates. Each thread will independently execute 'insert records where not exists' code (anti-semi join). Let's demo what that looks like.

Day 1 staged content

Hashed(ACCOUNT_ID)	Hashed(OTHER_ACCOUNT_ID)	Load-Date
hashed(1)	hashed(2)	1-Jan

Day 1 hub after staged content has loaded

Hashed(ACCOUNT_ID)	ACCOUNT_ID	Load-Date
hashed(1)	hashed(1)	1-Jan
hashed(2)	hashed(2)	1-Jan

Day 2 staged content, account id and other account id are the same

Hashed(ACCOUNT_ID)	Hashed(OTHER_ACCOUNT_ID)	Load-Date
hashed(3)	hashed(3)	2-Jan

Day 2 hub after staged content has loaded

Hashed(ACCOUNT_ID)	ACCOUNT_ID	Load-Date
hashed(1)	1	1-Jan
hashed(2)	2	1-Jan
hashed(3)	3	2-Jan
hashed(3)	3	2-Jan

We have duplicates because each loader acts independently! To illustrate this let's now try to load with a previously seen business key

Day 3 staged content, account id and other account id are the same but exists in the hub table

Hashed(ACCOUNT_ID)	Hashed(OTHER_ACCOUNT_ID)	Load-Date
hashed(1)	hashed(1)	3-Jan

Day 3 hub after staged content has loaded, no change

Hashed(ACCOUNT_ID)	ACCOUNT_ID	Load-Date
hashed(1)	1	1-Jan
hashed(2)	2	1-Jan

Hashed(ACCOUNT_ID)	ACCOUNT_ID	Load-Date
hashed(3)	3	2-Jan
hashed(3)	3	2-Jan

Day 4 finally, to further illustrate this concept, we will have staged duplicates in the account_id column.

Hashed(ACCOUNT_ID)	Hashed(OTHER_ACCOUNT_ID)	Load-Date
hashed(4)	hashed(5)	4-Jan
hashed(4)	hashed(6)	4-Jan

And now what happened? Recognize that the code to check against the target hub table must execute a select DISTINCT, however since the select distinct of the columns in staging is already unique the code will load BOTH account_id 4s.

Day 4 hub after staged content has loaded, more duplicates

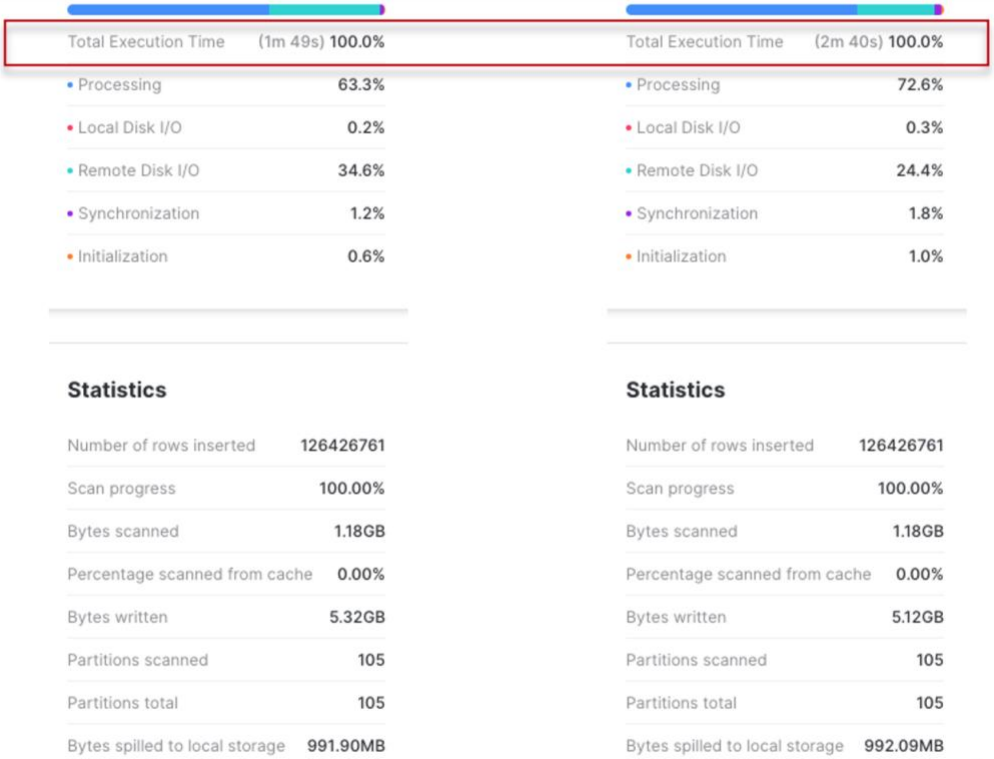
Hashed(ACCOUNT_ID)	ACCOUNT_ID	Load-Date
hashed(1)	1	1-Jan
hashed(2)	2	1-Jan
hashed(3)	3	2-Jan
hashed(3)	3	2-Jan
hashed(4)	4	4-Jan
hashed(4)	4	4-Jan
hashed(5)	5	4-Jan
hashed(6)	6	4-Jan

Therefore, multi-table inserts should not be used for loading data vault hubs, links, and satellites. It does have its place in loading point-in-time (PIT) tables however, which you can refer to that article here, see: bit.ly/3iEkBJC

3: Using Semi-structure functions on Structured columns for concatenation.

It might seem trivial but if data load volumes are large than picking the wrong function to perform concatenation before hashing becomes an expensive task!
Semi-structure functions used to perform column concatenation for creating a record hash **will** be slower than using the standard concatenation function available to structured data. Using the function might be pleasing to the eye, but slower in execution, observe.

DURATION	BYTES SCANNED	ROWS	QUERY TAG
42ms	—	—	ARRAY-TO-STRING semi-structure functior
2m 40s	1.2GB	126426761	ARRAY-TO-STRING semi-structure functior
44ms	—	—	CONCAT structure function
45ms	—	—	CONCAT structure function
1m 49s	1.2GB	126426761	CONCAT structure function



execution using structured column function

execution using semi-structured column function

You can start to see a difference by nearly 60 seconds, is that significant for your workload? Well, by second billing with a minimum of 60 seconds billed you might. Using the right function for the right data type.

Structured concatenation function:

```
 , sha1_binary(UPPER(concat(dv_tenantid, '||', dv_bkeycolcode_hub_account, '||', coalesce(to_char(trim(account_id)), '-1')))) as dv_hashkey_hub_account
```

Semi-structured concatenation function:

```
, sha1_binary(UPPER(ARRAY_TO_STRING(ARRAY_CONSTRUCT(dv_tenantid, dv_bkeycolcode_hub_account, coalesce(to_char(TRIM(account_id)), '-1')), '||'))) AS dv_hashkey_hub_account
```

4: Performing Hashing operations in the Loaders.

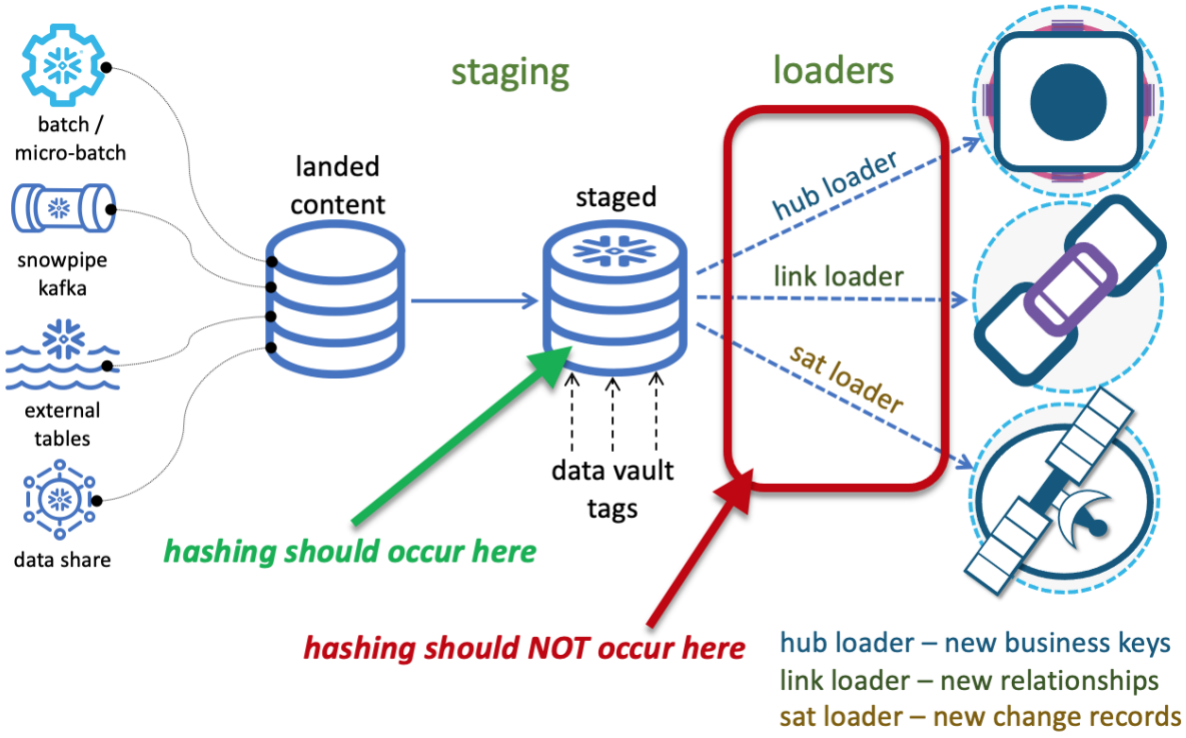
The automated process for getting data into a data vault is a combination of ***autonomous*** functions; each function has a ***single purpose*** driven by configuration. Once a file / table is landed it is then staged. The act of staging adds **DV-Tags** that are (amongst others):

- DV_LOADDATE
- DV_APPLIEDDATE
- DV_RECORDSOURCE
- DV_**HASHKEY**_<HUB-TABLENAME | LINK-TABLENAME>
- DV_HASHDIFF

The staged content (can be delivered as a view) is then used as a source for the loaders, and there are three loader-types:

- Hub loaders – ensures that it will only load new business keys and uniquely
- Link loaders – ensure that it will only load new relationships and uniquely
- Satellite loaders – ensure it only loads new changes and uniquely

What is the big deal?
Hash keys that are loaded to a hub-satellite has a parent hub table, if you are calculating the hash in the loaders for hub-satellite table AND the hub table then you're calculating the hash twice although they will be the same value! What happens if you're loading a link table that has two or more parent hub tables…?



Code for these tests is available here: bit.ly/3nyYF48

Test was executed by switching of result cache and flushing the XSMALL virtual warehouse after each execution.

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.